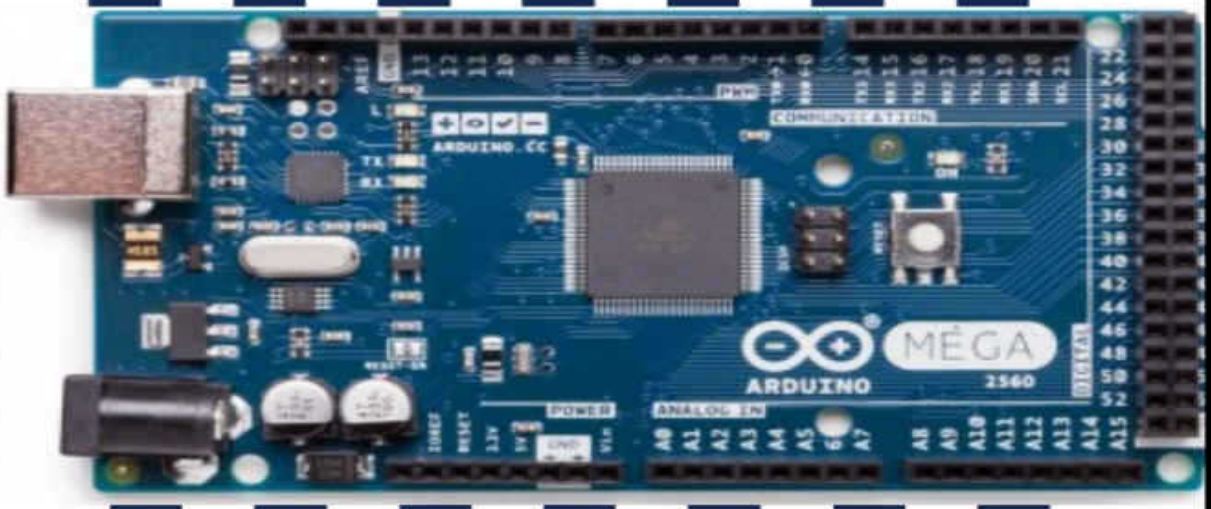
freeRTOS

Max Back



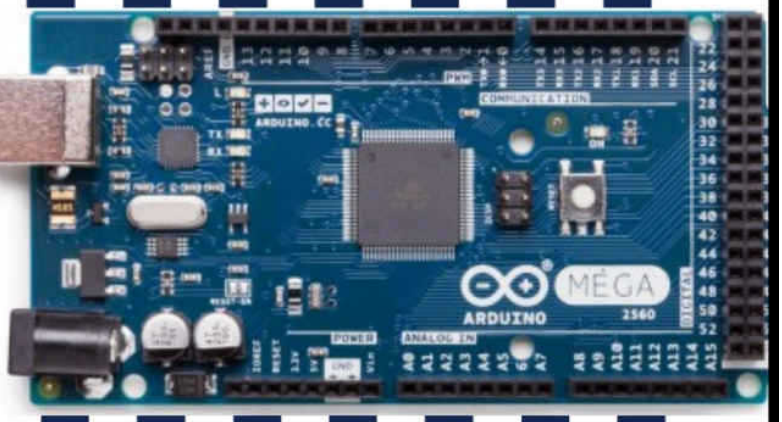
(Programando Multitarefa na prática - Utilizando a linguagem C/C++, freeRTOS e Arduino - Segunda Edição)

FREERTOS

PROGRAMACIÓN MULTITAREA EN LA PRÁCTICA - USANDO EL LENGUAJE C C++ Y FREERTOS ARDUINO

freeRTOS

Max Back



(Programando Multitarefa na prática - Utilizando a linguagem C/C++, freeRTOS e Arduino - Segunda Edição)

FREERTOS

PROGRAMACIÓN MULTITAREA EN LA PRÁCTICA - USANDO EL LENGUAJE C C++ Y FREERTOS ARDUINO

Programación Multitarea en la Práctica
Usando el lenguaje C/C++, freeRTOS e Arduino

Max Back

Prefácio

Este libro asume que tienes un conocimiento básico de al menos C y un lenguaje Arduino deseable (aunque hay una gran cantidad de material en línea).

Esta es una traducción de la segunda edición en portugués. Considere usar la versión original o la versión en inglés.

No recomendaré el uso de sistemas operativos aquí para permitir la multitarea sobre el desarrollo clásico. Cada uno tiene su propia declaración y particularidades.

Es posible que ya esté convencido de que necesita esta función o que desee conocerla. Por lo tanto, limitaré la cuestión de cómo usar esta función desde el principio, siempre que tenga conocimiento de C y, finalmente, de Arduino.

Algún tiempo después de su aparición (no sé exactamente) escuché sobre Arduino, y lentamente comencé a usarlo como una herramienta para jugar en las vacaciones y luego en el trabajo universitario. Finalmente llegué al punto de desarrollar un prototipo de producto en mi propia empresa. Siempre lo usé para la facilidad de obtener *shields* con circuitos adicionales, o incluso para montar placas con circuitos adicionales sobre placas de circuitos universales.

O freeRTOS foi a última peça do quebra-cabeça. Estava diante da decisão de conhecer e usar algum sistema operacional e entre Linux Embarcado e freeRTOS. Decidi aprofundar e conhecer o segundo, por ser menor, mais simples, compatível com circuitos de baixo *footprint* , ou seja, sem grandes necessidades de memória e MCU, basicamente.

Este libro es el resultado de mi esfuerzo de conocimiento, es mi contribución para popularizar el uso de sistemas operativos en tiempo real y la plataforma Arduino como ayuda para el desarrollo de productos innovadores.

En la primera parte presentaremos varios conceptos, recursos y ejemplos de uso de freeRTOS, buscando consolidar la comprensión de su uso y permitir si este RTOS será, desde este momento, un amigo que lo ayude en sus proyectos.

En la segunda parte del libro presento un proyecto un poco más grande destinado a hacer que las cosas malas sean interesantes, por así decirlo.

Este proyecto también se usa, pero con otro enfoque, en el libro "FreeRTOS como base para la programación multiplataforma: con Arduino y STM32" para ejemplificar el uso de freeRTOS como una herramienta para la programación multiplataforma también.

Poner en una sola aplicación la mayoría de las características presentadas. Este es mi proyecto CCT adaptado para este libro, que lee un sensor giroscópico externo y lo envía a un servidor a través de HTTP utilizando la tarjeta Expessif ESP8266 en modo de comando AT a través de Arduino serial1.

Qué es este libro: se puede describir y entender como un conjunto de ejemplos que muestran y explican el uso de las funciones freeRTOS en Arduino.

Una advertencia: gran parte de lo que voy a mostrar es del material de documentación de freeRTOS, que puede (y debe) estudiarse para su posterior estudio. Acceda a este material en https://www.freertos.org/Documentation/RTOS_book.html .

El valor de este libro es que muestra el uso de la multitarea y explica los pasos, lo que lleva al lector a aprender y tener el entorno Arduino. Puede aprovechar este conocimiento en otros entornos.

La traducción al español es asistida por herramientas de traducción. Si encuentra algún problema serio, hágamelo saber.

¡Les deseo una buena lectura y un buen aprendizaje!

Max de vuelta
Ingeniero Electricista y
Programador de Sistemas Embebidos e Información

Resumen

Guia de este libro

Estructura

Este libro está diseñado para guiar al estudiante a través de un ritmo de conocimiento incremental, mezclando código y explicaciones, con una invitación constante a la programación (práctica, como dicen).

Los fragmentos de código, los listados famosos, se presentarán en texto plano, con fuente fija, acompañados por los números de línea a la izquierda (lo que hace que sea mucho más fácil que una línea o rango de línea sea referenciado por su número). Aquí hay un ejemplo:

Listado de ejemplo:

```
01: #include <Arduino_FreeRTOS.h>
02: //Comentario
03: void setup()
04: {
05:   Serial.println(“Ola Arduino e freeRTOS!\n”);
06: }
```

Así que podría comentar, a modo de ejemplo, que en las líneas 4 y 6 podemos ver una llave de apertura y cierre ({y}). Por supuesto, no debe escribir estas líneas en sus archivos de origen.

Sé que esto hace que sea difícil copiar y pegar código si está viendo este libro en una PC, por ejemplo.

Los comentarios están en cursiva y gris, como se muestra en la línea 2, para resaltar el código en sí, y se insertarán cuando se considere relevante para la lectura del código.

Tratando de abordar este problema, al menos en parte, los listados más completos se adjuntarán al final del libro, sin los números de línea, para que pueda copiarlos al IDE de Arduino.

Fuentes públicas en github

Algunos de los códigos más básicos también se pueden encontrar en github. Siempre que el logotipo de github (imagen de arriba) esté cerca del comienzo de la lista (o menos comúnmente en otra parte del texto) significa que esta lista está disponible en github en la siguiente dirección:

https://github.com/maxback/multitarefa_na_pratica

Primera parte: instalación, conceptos y ejemplos

En esta primera parte, instalamos el entorno en su IDE de Arduino y trabajamos en conceptos y ejemplos de usos de las funciones freeRTOS que admiten estas características.

Como nuestro enfoque es un esfuerzo por la practicidad, trataremos de abordar sucintamente los conceptos y ejemplos, pero no incluirán una gran cantidad de periféricos y características de hardware, centrándonos más en los pines de entrada y salida y enviando a través de serie conectada a la PC. a través de USB.

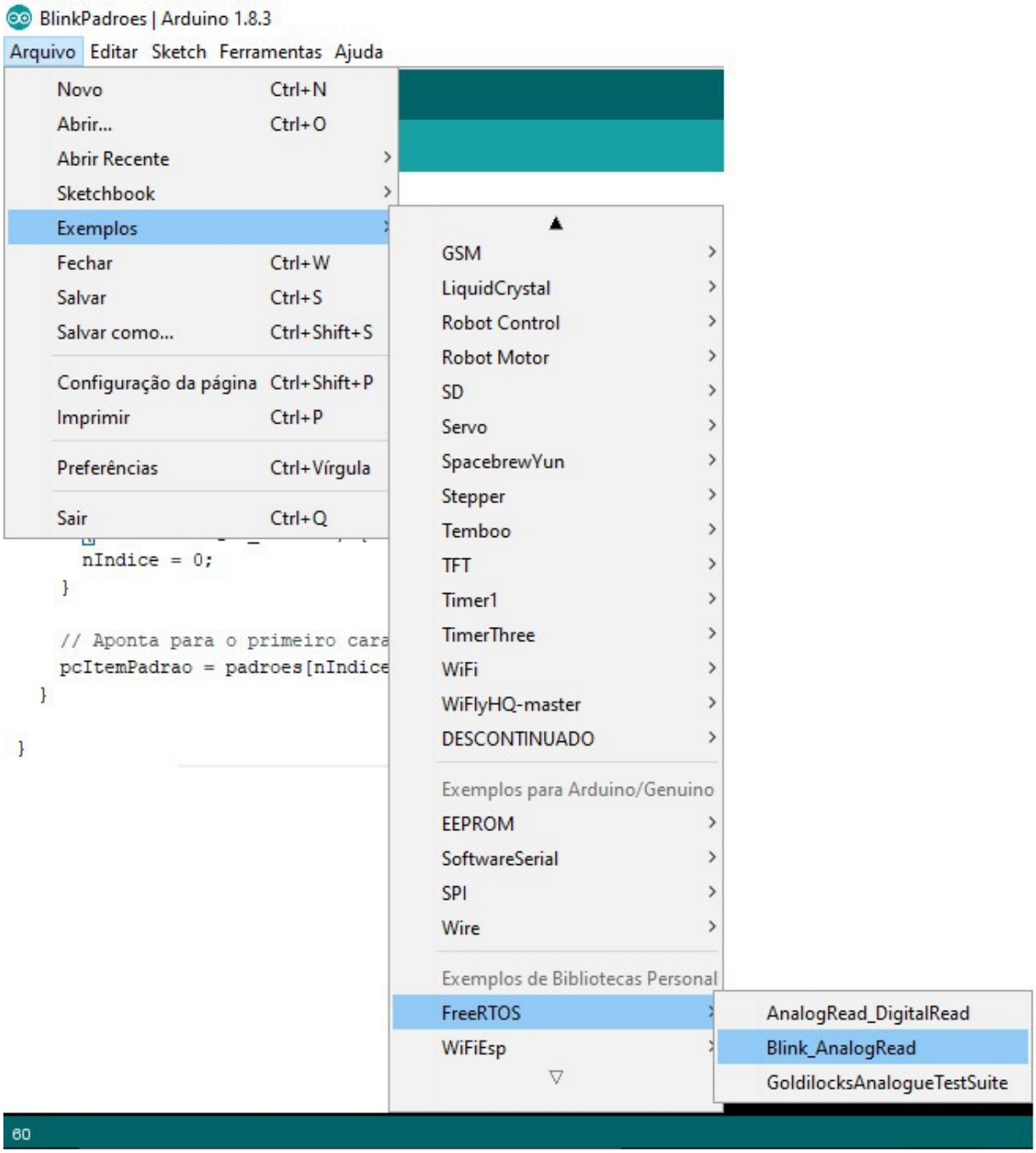
Capítulo 1 - Comenzando con freeRTOS para Arduino

Nota: ya debe tener instalado Arduino IDE. Si este no es el caso, puede obtenerlo en la siguiente dirección:

https://www.arduino.cc/en/Main/Software

Elija la versión IDE más apropiada e inicie la descarga. Solo instálalo. A continuación, debe instalar una biblioteca Arduino que incluya todos los freeRTOS. Como este procedimiento es un requisito previo, la instalación paso a paso se puede ver en el Apéndice F - Instalación de la biblioteca freeRTOS .

En el IDE de Arduino, cargaremos un proyecto de muestra freeRTOS, que viene con la biblioteca freeRTOS. Ya hemos verificado que está presente en el IDE: vaya al menú Archivo \ Ejemplos \ FreeRTOS \ Blink_AnalogRead:



Listo!

Se debe abrir un proyecto, con Arduino lib include y dos tareas, una para parpadear el LED de la placa Arduino y otra para enviar lecturas desde una de las entradas analógicas a través de la serie.

Primeras líneas de archivo Blink_AnalogRead.ino:

```
01: #include <Arduino_FreeRTOS.h>
02:
03: // define two tasks for Blink & AnalogRead
04: void TaskBlink( void *pvParameters );
05: void TaskAnalogRead( void *pvParameters );
```

Estas son solo las primeras líneas del archivo de código abierto. Volveremos a ello pronto (puede dejarlo abierto si lo desea), pero primero debemos cubrir algunos minutos de los conceptos básicos del próximo capítulo, que pueden ser útiles para comprender mejor algunos términos que se utilizarán en este libro.

En los próximos capítulos trabajaremos a partir de ejemplos estándar de Arduino y la biblioteca instalada, creando nuevos proyectos desde cero después de eso.

Capítulo 2 - Conceptos básicos del sistema operativo en tiempo real

Sistemas operativos multitarea

Hay sistemas operativos únicos y multitarea. Un ejemplo de un sistema operativo de una sola tarea es el antiguo MS-DOS de Microsoft. Había un sistema operativo, ya que su programa no se ejecutó directamente en el hardware, ya que hacemos un *firmware* (o *baremetal*). De hecho, hay un sistema operativo que desempeña el importante papel intermedio entre el complejo hardware de una computadora y nuestro programa.

Sin embargo, a diferencia de Windows, Linux e incluso el sistema operativo de nuestros *smartphones* , solo permitía una aplicación de usuario a la vez. Incluso podemos escribir algún software creando *driver* o *device drivers* , con el objetivo de manejar algunas funciones de bajo nivel (como una tarjeta instalada en su PC o un puerto serie), incluido el uso de interrupciones de hardware o sistema operativo, en "paralelo" a programa en ejecución

Los sistemas multitarea le permiten iniciar más de un programa o tarea a la vez. Si solo hay un procesador y en este procesador solo hay un núcleo, estos programas no se ejecutarán simultáneamente, sino que alternarán el tiempo de CPU, que ejecuta el tiempo de un programa, el tiempo de otro. Para el usuario es la ilusión de que todo se está haciendo en paralelo, realizando tareas como descargar un programa de Internet y guardarlo en el disco, reproducir música desde el disco duro y recibir la entrada del teclado y el mouse en un editor de texto, mientras Recibirá una notificación de que acaba de recibir un mensaje instantáneo de uno de sus contactos.

Si hay múltiples núcleos y procesadores, los programas pueden ejecutarse en paralelo, pero no todos a la vez, obviamente.

Tarea (*task*)

Las tareas (*tasks)* , como freeRTOS, son el equivalente de sus programas informáticos. Pensemos en ello de forma específica para la programación de la consola C: el típico programa universitario C / C ++ es uno que tiene la función `main()` en su interior y tiene un *loop* infinito que le pide al usuario opciones, respondiendo cada opción ejecutando un

fragmento de código específico y luego volviendo a una nueva interacción de *loop* , que nuevamente solicita una nueva opción.

Consejo: Notará que el fragmento de programa usa las funciones de Arduino, tanto para configurar los pines como para otras funciones. Si nos detenemos para notar, este programa separa la parte de configuración al comienzo de la función `main()` y el cuerpo de la ejecución que se recibe dentro del ciclo for (;;). Los usuarios de Arduino no tienen acceso a la función `main()` . Siempre implementa las funciones `setup()` y `oop()` , que en la práctica es tener una forma de acceder a la ejecución de arranque y desde dentro del bucle principal del programa Arduino sin tener que ver los detalles de bajo nivel de la plataforma.

Listado 1 - Ejemplo de una aplicación de tarea única de microcontrolador C

```
01: void main(void) {
02:
03:    //Configure el pin 13 como salida (LED) y 1 como entrada

04:   pinMode(13, OUTPUT);
05:   pinMode(1, INPUT);
06:
07:    //Comienza en serie a 9600 bps
08:   Serial.begin(9600);
09:
10:    //lazo principal
11:   for(;;) {
12:     digitalWrite(13, HIGH);
13:     delay(1000);
14:     digitalWrite(13, LOW);
15:     delay(1000);
16:
17:      //Enviar estado del interruptor de entrada al pin 1
```

```
18:     if(digitalRead(1) == HIGH) {
19:       Serial.println("¡La llave ha sido activada!");
20:     } else {
21:       Serial.println("¡La llave ha sido apagada!");
22:     }
16:   }
17: }
```

El punto a tener en cuenta es que este programa hace dos cosas (después de que todo está configurado):

- El LED en el puerto 13 parpadea.
- Enviar por serie el estado del botón (activar o desactivar).

Estas dos tareas a realizar, una vez, otra vez, se realizan incluso si se trata de un solo programa. Para resolver el problema buscamos algún tipo de solución. Esta solución presentada es un buen punto de partida, ya que busca encontrar una manera de hacer lo que se necesita.

Podríamos decir que el código entre las líneas 12 y 15 es una de las tareas, mientras que el código entre las líneas 18 y 22 corresponde a otra tarea.

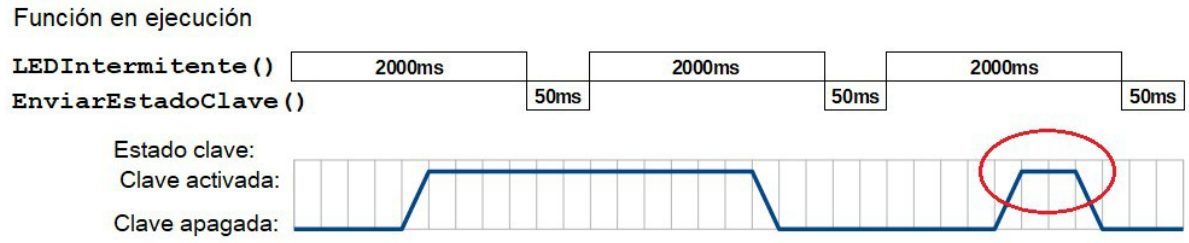
Si creáramos dos funciones, cada una para encapsular el código para una de estas tareas, tendríamos un código más organizado. Aquí está el código rehecho, que muestra ambas funciones en negrita, así como sus llamadas:

Listado 1b - Aplicación de tarea única modificada

```
01: void piscaLED(void) {
02:   digitalWrite(13, HIGH);
03:   delay(1000);
04:   digitalWrite(13, LOW);
05:   delay(1000);
06: }
07:
08: void enviaEstadoChave(void) {
09:   if(digitalRead(1) == HIGH) {
```

```
10:     Serial.println("¡La llave ha sido activada!");
11:   } else {
12:     Serial.println("¡La llave ha sido apagada!");
13:   }
14: }
07:
01: void main(void) {
02:
03:   //Configure el pin 13 como salida (LED) y 1 como entrada

04:   pinMode(13, OUTPUT);
05:   pinMode(1, INPUT);
06:
07:   //Comienza en serie a 9600 bps
08:   Serial.begin(9600);
09:
10:   //lazo principal
11:   for(;;) {
16:     piscaLED();
17:     enviaEstadoChave();
16:   }
17: }
```

Na listagem 1b temos uma melhor organização do código, porém continuamos com as características de execução. Da forma como está o programa, uma mudança na chave, seguida da reversão desta mudança (ligar e desligar em seguida) não seria percebida, se ocorresse enquanto está piscando o LED, ou seja, enquanto está na función `piscaLED()` .



En este ejemplo, suponemos que el LED de función `LEDIntermitente()` tarda 2000 ms (dos segundos) y la función `EnviarEstadoClave()` 50 ms.

Durante la tercera llamada a la función `LEDIntermitente()` , la tecla se enciende y se apaga, lo que hace que se ignore el evento, por lo que el

texto enviado por el sería el siguiente:

```
¡La llave ha sido activada!
¡La llave ha sido apagada!
¡La llave ha sido apagada!
```

Sin embargo, lo que realmente debería haberse enviado al usuario sería lo siguiente (la diferencia está subrayada):

```
¡La llave ha sido activada!
¡La llave ha sido apagada!
¡La llave ha sido activada!
¡La llave ha sido apagada!
```

La solución para mantener un evento detectable, incluso si se realiza una tarea que no sea una lectura directa de su valor, es usar una interrupción de hardware, donde podría cambiar el valor de una variable global que luego se leería en la función. Las interrupciones son tipos de tareas que le permiten escribir (lo más corto posible) el código que se ejecutará cuando se detecte un evento, como un cambio en el estado de un pin de entrada. En cierto modo, es un paso hacia la programación multitarea.

El uso de interrupciones no se elimina al programar con un RTOS, pero se hace más fácil ya que podemos usar mecanismos más robustos para "hablar" entre interrupciones y tareas escritas para manejar estos datos y eventos. Por lo tanto, mecanismos manuales y potencialmente peligrosos, como una variable global, que deben probarse en el *loop* principal

A continuación se incluye un pseudocódigo que muestra la función de manejo de cambios clave y otra de una tarea que maneja este valor. La configuración del hardware y los detalles de la rutina de interrupción se omiten:

Listado 2 - Pseudocódigo de una tarea junto con una interrupción

```
01: void trataInterrupcionPinEntrada(void) {
02:    //Enviar información de estado clave
```

```
03:   EnviarInformacionALaTask(digitalRead(1));
04: }
05:
06: void taskEnviaEstadoClave(void) {
07:
08:   //Variable local para leer información
09:   char EstadoClave;
10:
11:    //loop principal de tareas
12:   for(;;) {
13:      //Espera recibir alguna información para eltask
14:     EstadoClave = RecibirInformacionEnLaTask();
15:
16:     if( EstadoClave == HIGH) {
17:       Serial.println("¡La llave ha sido activada!");
18:     } else {
19:       Serial.println("¡La llave ha sido apagada!");
20:     }
21:   }
22: }
```

Tenemos entre las líneas 1 y 3 la definición de una función de manejo de interrupciones, que se llamará cada vez que cambie el pin conectado a nuestra tecla. Esta rutina solo lee el nuevo valor clave y llama a una función de soporte multitarea (ficticia) para enviar este valor a la tarea denominada `EnviarInformacionALaTask()` , que comienza en la línea 6.

De esta manera, la interrupción dura lo más breve posible, volviendo al punto de código anterior, por ejemplo. Cuando llegue el momento de realizar la tarea, recibirá la información con la función `RecibirInformacionEnLaTask()` , también ficticia. Luego decida qué texto enviar por serie.

La función de *task* se parece a una función `main()` , con un *loop* infinito en su núcleo, que muestra que se ejecuta sin tener que molestarse en cooperar con otras funciones y tareas que puedan existir. Será el propio sistema operativo el que momentáneamente dejará de ejecutar el código de la *task* para darle una oportunidad a otra *task* . Esto puede suceder de vez en cuando, así como cuando la *task* actual espera algo que no ha llegado, por un tiempo o de forma indefinida.

En la línea 14, la llamada para recibir información es una llamada configurada para esperar indefinidamente. Entonces, esta *task* se congela

hasta que se le envía algo, que en nuestro caso es cuando el estado de la clave cambia y la interrupción maneja el cambio.

La programación está mucho más localizada de esta manera porque la *task* tiene una interfaz bien definida con el mundo exterior: interrupciones, pines de I/O, otras *task* , etc. Al observar esta tarea, podemos ver que espera cada información, sin hacer nada hasta que llegue algo. Luego envía el texto apropiado al serial y la siguiente interacción de *loop* regresa para esperar nuevamente, y así sucesivamente.

En otros casos más complejos, las funciones RTOS se pueden usar con la opción de esperar algún tiempo para obtener información, escribiendo un tratamiento adecuado en caso de que no ocurra nada, es decir, no llegan datos en este rango.

También podemos realizar *tasks* que simplemente hagan algo no relacionado con el resto del programa, como parpadear el LED en nuestro ejemplo anterior. En este caso, la *task* se puede ejecutar desde el principio solo para indicar que la tarjeta tiene algún software en ejecución. Podríamos cambiar esta *task* para recibir cambios en el valor de *delay* para que los LEDS parpadeen, por ejemplo, la frecuencia de parpadeo.

De hecho, podemos dividir varias tareas de nuestros sistemas en *task* y organizar el intercambio de datos con mecanismos del sistema operativo como semáforos y colas para que la ejecución de cada parte del programa se realice en el momento adecuado.

Todavía existe el problema de las prioridades de las *task* , de modo que las *task* de mayor prioridad tienen más probabilidades de desempeñarse que las de menor prioridad. Por lo tanto, cada cambio de *task* que se realiza analiza la *task* que se debe realizar a continuación. Si una *task* con mayor prioridad que las demás está lista para ejecutarse (porque la información esperada ha llegado, por ejemplo), se ejecutará antes que las demás. Incluso hay mecanismos para realizar la *task* que se estaba ejecutando cuando se produjo una interrupción, no necesariamente la *task* que continuará después de la interrupción.

Imagine que nuestro cambio es solo una de varias *tasks* y que, en el momento de la interrupción de la tecla, se está ejecutando otra *task* (y continuaría durante algunas fracciones de segundo todavía). Bueno, si la *task* que maneja la clave tiene baja prioridad, es posible que no se ejecute a continuación, pero si su prioridad es la más alta, ciertamente se ejecutará justo después de la interrupción. Así podemos definir *tasks* donde la idea

del tiempo real es importante y otras donde no tanto. En general, esta es la idea detrás de los términos tiempo real duro (*hard real time*) y tiempo real suave (*soft real time*).

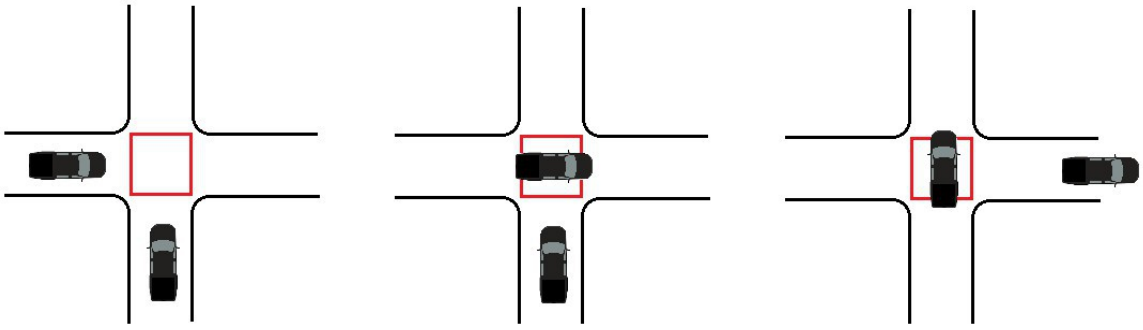
He tratado de cubrir superficialmente algunos conceptos que pueden entenderse mejor en la práctica a lo largo del libro y en la lectura de material adicional, como los manuales freeRTOS, cuyo enlace se agregó al comienzo de este libro.

Semáforo (Semaphore)

Los semáforos son sistemas de señalización que permiten coordinar la ejecución entre más de una *task* . Es a través de ellos que las *tasks* pueden realizar señales, sin necesariamente tener datos asociados, solo el contexto asociado por el programador con el propio semáforo.

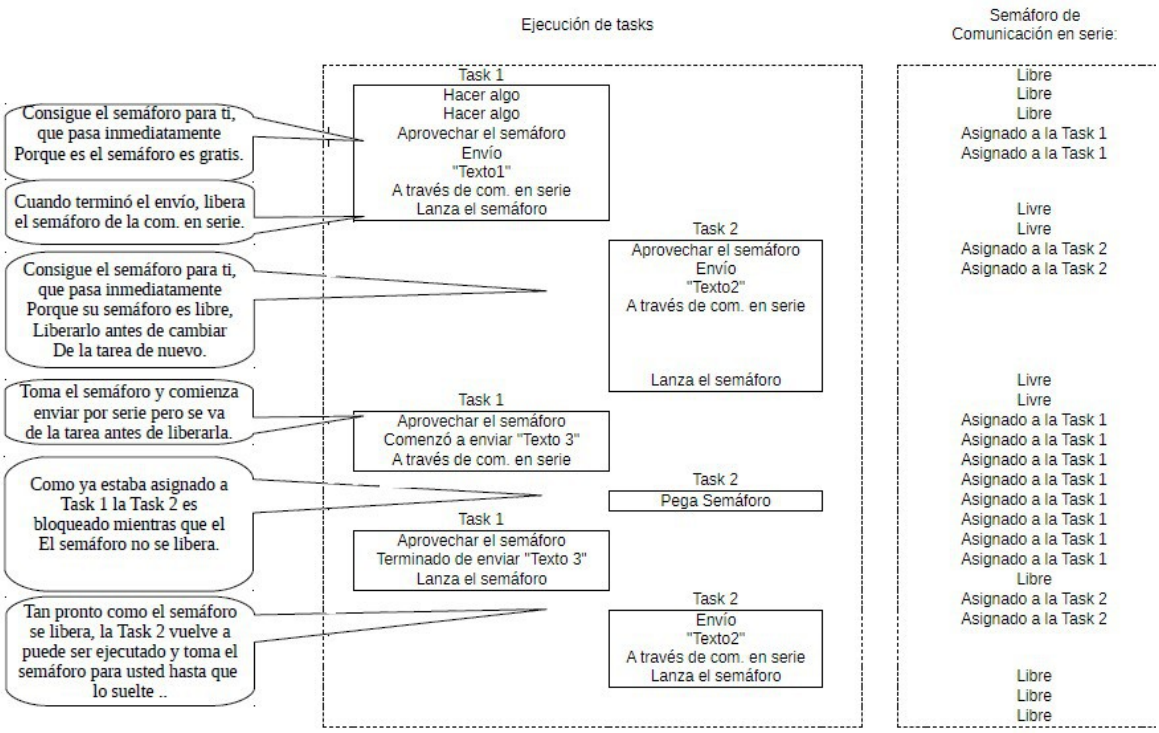
Un tipo de semáforo es mutuamente exclusivo, lo llamamos MUTEX (MUTual EXclusion, en inglés), es decir, señalan como un contexto en el que solo una de las *tasks* puede actuar, y que todas las demás tienen la intención de se excluye actuar en ese contexto, como una intersección de calle con un semáforo.

En este ejemplo, el hecho de que un sentido pase a través de la intersección excluye la posibilidad de que el otro sentido pase allí también. En un segundo instante, la segunda dirección usará la intersección, lo que excluye la posibilidad de que los autos en la primera dirección se crucen y se produzca una colisión.



Este objeto es conocido por todas las *tasks* interesadas en usar este contexto, y tiene controles de concurrencia al acceder al objeto garantizado por la implementación del sistema operativo, lo que garantiza (como el único que altera el estado del semáforo a través de su llamada) que Dos tareas no tomarán contexto para ellos simultáneamente, causando colisiones.

La siguiente imagen ilustra el uso de un semáforo MUTEX para controlar el acceso de más de una tarea a las funciones de envío en serie de arduino.



Podemos notar que el semáforo se está asignando a ambas *taskas* . Cada *task* utiliza la función de serie a la vez, liberando el semáforo cuando ya no necesita comunicación serial, permitiendo así que la otra *task* (o ella misma) la asigne en el futuro.

En algún momento (en la segunda ejecución de la *task* 2) el semáforo intenta ser asignado, pero ya está hecho, lo que hace que la *task* que intentó asignar se bloquee hasta que se libera el semáforo.

Nota: Elegir (asignar) y liberar (desasignar) el semáforo puede bloquear la *task* indefinidamente (como en este ejemplo), o hasta que ocurra un cierto *timeout* de espera definido por el programador, y se debe manejar un retorno de error en la función de llamada. el semáforo, decidiendo qué hacer a continuación.

Además de estos semáforos de exclusión mutua (MUTEX) o semáforos binarios, tenemos semáforos de contador, que permiten que varias zero y puntos de código asignen el semáforo, disminuyendo un valor hasta que llegue a cero, cuando ya no será posible asignarlo. . Las operaciones de desasignación o liberación incrementan este contador nuevamente. A lo largo del free se presentarán ejemplos de los dos tipos de semáforos, aclarando de manera práctica su funcionamiento.

Sección crítica

Una instrucción C suele ser equivalente a varias instrucciones de lenguaje de máquina. Por lo tanto, cuando utilizamos el acceso a la variable global, dentro de nuestras *task* tenemos un caso en el que no se puede garantizar que la tarea deje el contexto con la instrucción a la mitad y otra *task* tomará la variable con un valor inconsistente.

Cualquier operación que deba realizarse de principio a fin, sin cambiar la *tasks* actual a otra, puede realizarse dentro de una sección de código protegido, donde el mecanismo de cambio de *tasks* está desactivado (y se activará solo al final del bloque). Este bloque es lo que llamamos la sección crítica.

La sección crítica comienza con una instrucción especial del sistema operativo, seguida de la instrucción (o instrucciones) que deben protegerse de la concurrencia para finalizar la sección crítica con otra instrucción del sistema operativo. En el caso de freeRTOS se llama un par de macros.

Tiempo(software *timer*)

Además del código que ejecuta *tasks* , podemos usar funciones que se ejecutan de vez en cuando sin un *loop* interno desde el módulo del *timer* . El módulo del *timer* coloca las funciones definidas en una cola y las realiza a intervalos regulares o una vez después de un temporizador.

Por lo tanto, para parpadear un LED o controlar un *timeout* de espera, se puede activar una llamada de función cada n segundos, por ejemplo. Si ve que ya no tiene que esperar eso (porque la información que espera ha llegado, por ejemplo), simplemente llame a una función RTOS para cancelar la configuración de esta función.

Capítulo 3 - Trabajando con tasks

Usando los ejemplos que vienen con la biblioteca

Reanudemos ahora esa fuente de muestra que está disponible en el IDE de arduino en la ruta del menú: Archivo \ Ejemplos \ FreeRTOS \ Blink_AnalogRead.

La fuente no se enumerará aquí por completo, pero algunos de sus extractos se resaltarán y, sin embargo, sin ningún comentario. Puede buscarlos por secuencia en su IDE familiarizándose con las partes del programa Arduino con freeRTOS. Luego veremos otro ejemplo y se darán explicaciones más detalladas. Las referencias de línea son útiles, pero no hay garantía de que su ejemplo IDE no difiera del mío cuando abrí este archivo.

Incluyendo el encabezado (header) freeRTOS predeterminado para arduino

Este es un comando de inclusión mínimo estándar para usar freeRTOS en arduino. Tiene las definiciones más básicas, que permiten la creación y operación de *tasks* .

Es la primera línea del archivo:

```
01: #include <Arduino_FreeRTOS.h>
```

Si se utilizan otros recursos, como los semáforos, se requerirá la inclusión de otros archivos de encabezado de acuerdo con cada situación.

Prototipo de funciones del tasks

Para que possa ser referenciada na função `setup()` , que cria tarefas, seus protótipos são definidos da seguinte forma:

```
03: // define two tasks for Blink & AnalogRead

04: void TaskBlink( void *pvParameters );

05: void TaskAnalogRead( void *pvParameters );
```

Tus nombres deben ser que tu reflejo es tu objetivo. Siempre devuelva `void` y reciba un puntero genérico a `void` (que se puede adaptar o al tipo de

destino que está destinado a pasar a una *task* , como veremos más recientemente).

Creación de taks

En la función `setup()` , ambas *tasks* se crean utilizando la función freeRTOS `xTaskCreate()` . Cada función tiene algunos parámetros, y por ahora solo el primero, que es el nombre de la función de tarea. Entonces freeRTOS crea todos los controles de *tasks* y prepara el sistema para ejecutar la función pasada por parámetros como el código de *tasks* :

```
17: // Now set up two tasks to run independently.
18:   xTaskCreate(
19:     TaskBlink
20:     ,  (const portCHAR *)"Blink"
21:     ,  128
22:     ,  NULL
23:     ,  2
24:     ,  NULL );
25:
26:   xTaskCreate(
27:     TaskAnalogRead
28:     ,  (const portCHAR *) "AnalogRead"
29:     ,  128  // Stack size
30:     ,  NULL
31:     ,  1  // Priority
32:     ,  NULL );
```

En negrita, la función se pasa con el código de *task* .

Función loop vacío

Puede encontrar que la función `loop()` , que es donde puede esperar encontrar el código que parpadea el LED y lee una entrada analógica, está vacía. Esta es precisamente la razón por la cual ahora las operaciones estarán preferiblemente en *tasks* y cuando sea necesario en interrupciones.

Esta función sin nada evidencia, en mi opinión, sobre todo el cambio de paradigma de este tipo de programa. [1] .

Funciones de taks

Las siguientes son las implementaciones de las funciones, omitiendo bloques de comentarios:

```
46: void TaskBlink(void *pvParameters)  // This is a task.
47: {
48:   (void) pvParameters;
49:
74:   // initialize digital LED_BUILTIN on pin 13 as an output.
75:   pinMode(LED_BUILTIN, OUTPUT);
76:
77:   for (;;) // A Task shall never return or exit.
78:   {
79:     digitalWrite(LED_BUILTIN, HIGH);
80:     vTaskDelay( 1000 / portTICK_PERIOD_MS );
81:     digitalWrite(LED_BUILTIN, LOW);
82:     vTaskDelay( 1000 / portTICK_PERIOD_MS );
83:   }
84: }
```

En la línea 48, una referencia al parámetro, que en realidad no hace nada, sirve para eliminar una alerta de que el parámetro no se utiliza. No puede eliminarlo, ya que la firma siempre debe ser esta, siendo una de las soluciones para satisfacer el ***linker*** .

En la línea 75, tenemos una inicialización de pin. Una cosa interesante es que podemos hacerlo en las primeras líneas de la *task* , ya que este pin es su negocio y solo ella lo usará. Este código se ejecuta solo una vez antes de que la *task* entre en su *loop* eterno en la línea 77.

Entre las líneas 77 y 83 tenemos un bucle que parpadea el LED. Primero encienda el LED conectado al pin, espere un momento, apáguelo y vuelva a encenderlo. Esta es la posible implementación para lo que pretendía implementar sin usar un RTOS en la función intermitente () de Listado 1 - Ejemplo de una aplicación de tarea única de microcontrolador C .

Veamos ahora la segunda función de tasks:

```
086: void TaskAnalogRead(void *pvParameters)  // This is a
task.

087: {

088:   (void) pvParameters;


099: for (;;)

100: {

101:     // read the input on analog pin 0:

102:     int sensorValue = analogRead(A0);

103:     // print out the value you read:

104:     Serial.println(sensorValue);

105:     vTaskDelay(1);

106:   }
```

```
107: }
```

En la línea 102 tenemos una lectura de la entrada analógica A0 para la variable `sensorValue` que se asigna dentro del *loop* .

En la línea 104, este valor se envía por serie. Si compara los valores de serie enviados aquí con el código que se mostrará en el Listado 3: AnalogRead_DigitalRead (ejemplo de biblioteca sin algunos comentarios) , notará que no era necesario usar un semáforo para coordinar el uso exclusivo de serie aquí simplemente porque solo una *task* envía cosas por el serial.

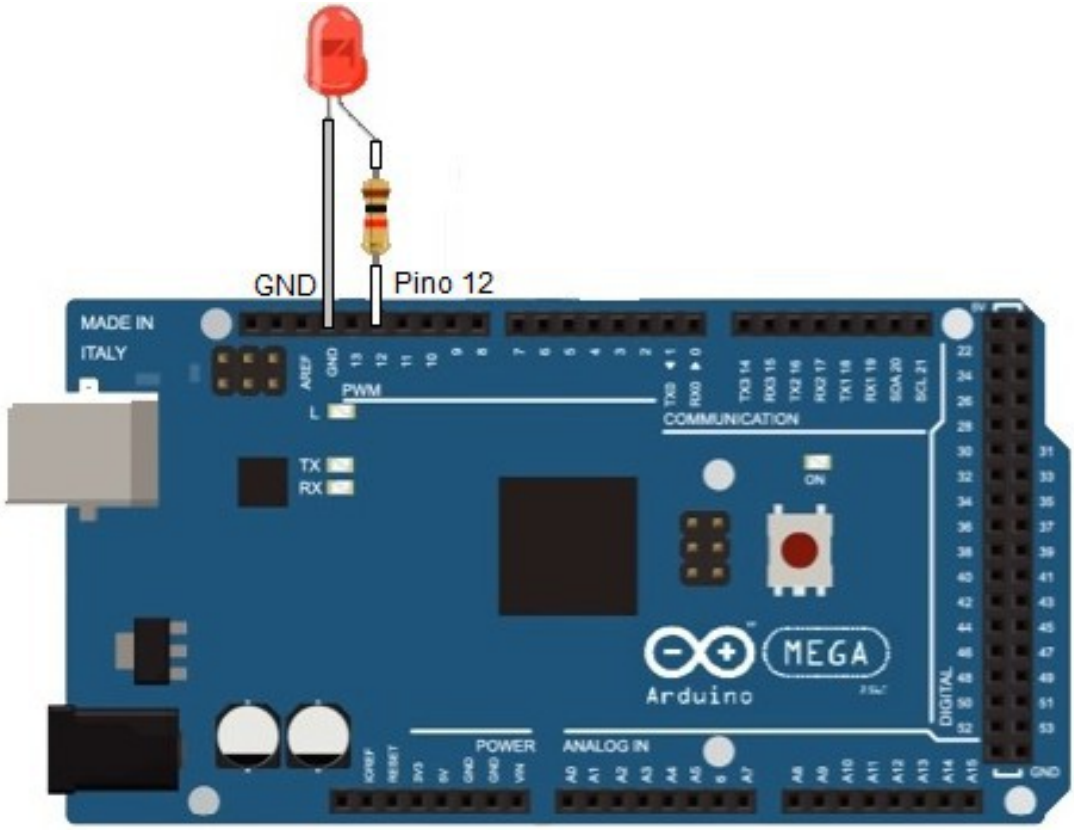
Si usa un recurso en una sola *task* , puede usarlo fácilmente, pero si lo usa en varios lugares, necesita controlar el acceso a él.

Al igual que la función `TaskBlink()` , esta función de *task* tiene un *loop* . Sin embargo, en este caso, no se requiere inicialización.

Es posible que desee ejecutar este ejemplo en su arduino y ver al monitor en serie enviar valores de entrada A0 (incluso si no le ha conectado ningún sensor). Si busca el LED en la placa Arduino (independientemente del modelo, existe este LED en la placa Arduino en sí, conectado internamente a la salida 13), verá que parpadea según lo previsto.

FreeRTOS se preocupa por compartir la CPU entre los códigos de las dos funciones, cambiándolas constantemente. Intente copiar la función `TaskBlink()` a otra función llamada `TaskBlinkSaida12` , por ejemplo, insertando su prototipo antes de la función `setup()` y creando una *task* más con el mismo código que la otra pero con el pin 12 parpadeando.

Si la idea suena bien, use un LED con el cátodo conectado al GND y una resistencia de 10kΩ que conecte el ánodo al pin 12, de la siguiente manera:



Observação:

Este código modificado se puede encontrar en github. En la dirección https://github.com/maxback/multitarefa_na_pratica

Un segundo ejemplo

Para reforzar los conceptos básicos del esqueleto del programa, veamos otro ejemplo. Se recomienda, especialmente desde el principio, utilizar esqueletos ya preparados como base para nuevos proyectos, ahorrando tiempo y evitando errores comunes, como olvidar incluir o asignar mal una función de *task* .

Abra una segunda fuente de muestra, también disponible en el IDE de Arduino desde la ruta del menú: Archivo \ Ejemplos \ FreeRTOS \ AnalogRead_DigitalRead.

El archivo abierto tiene varios comentarios explicativos. Algunos de estos comentarios han sido eliminados, pero aún resultan en un código relativamente largo presentado a continuación.

Listado 3: AnalogRead_DigitalRead (ejemplo de biblioteca sin algunos comentarios)

```
01: #include <Arduino_FreeRTOS.h>
02: #include <semphr.h>  // add the FreeRTOS functions for
Semaphores (or Flags).
03:
04: SemaphoreHandle_t xSerialSemaphore;
05:
06: // define two Tasks for DigitalRead & AnalogRead
07: void TaskDigitalRead( void *pvParameters );
08: void TaskAnalogRead( void *pvParameters );
09:
10: // the setup function runs once when[...]
11: void setup() {
12:
13:   // initialize serial communication at 9600 bits per
second:
14:   Serial.begin(9600);
15:
16:   while (!Serial) {
17:     ;
18:   }
19:
20:   if ( xSerialSemaphore == NULL )
```

```
21:   {
22:     xSerialSemaphore = xSemaphoreCreateMutex();
23:     if ( ( xSerialSemaphore ) != NULL )
24:       xSemaphoreGive( ( xSerialSemaphore ) );
25:   }
26:
27:   // Now set up two Tasks to run independently.
28:   xTaskCreate(
29:     TaskDigitalRead
30:   ,  (const portCHAR *)"DigitalRead"  // A name just for
humans
31:   ,  128  // This stack size can be checked & adjusted by
reading the Stack Highwater
32:   ,  NULL
33:   ,  2  // Priority, with 3 (configMAX_PRIORITIES - 1)
being the highest, and 0 being the lowest.
34:   ,  NULL );
35:
36:   xTaskCreate(
37:     TaskAnalogRead
38:   ,  (const portCHAR *) "AnalogRead"
39:   ,  128  // Stack size
40:   ,  NULL
41:   ,  1  // Priority
42:   ,  NULL );
43:
44:   // Now the Task scheduler, which takes over control of
scheduling individual Tasks, is automatically started.
45: }
46:
47: void loop()
48: {
49:   // Empty. Things are done in Tasks.
50: }
51:
52: /*--------------------------------------------------*/
53: /*---------------------- Tasks ---------------------*/
54: /*--------------------------------------------------*/
55:
56: void TaskDigitalRead( void *pvParameters
__attribute__((unused)) )  // This is a Task.
57: {
```

```
58:    // digital pin 2 has a pushbutton attached to it. Give it
a name:
59:   uint8_t pushButton = 2;
60:
61:    // make the pushbutton's pin an input:
62:   pinMode(pushButton, INPUT);
63:
64:   for (;;) // A Task shall never return or exit.
65:   {
66:      // read the input pin:
67:     int buttonState = digitalRead(pushButton);
68:
69:     if ( xSemaphoreTake( xSerialSemaphore, ( TickType_t ) 5
) == pdTRUE )
70:     {
71:       Serial.println(buttonState);
72:
73:       xSemaphoreGive( xSerialSemaphore );
74:     }
75:
76:     vTaskDelay(1);  // one tick delay (15ms) in between
reads for stability
77:   }
78: }
79:
80: void TaskAnalogRead( void *pvParameters
__attribute__((unused)) )  // This is a Task.
81: {
82:
83:   for (;;)
84:   {
85:      // read the input on analog pin 0:
86:     int sensorValue = analogRead(A0);
87:
88:     if ( xSemaphoreTake( xSerialSemaphore, ( TickType_t ) 5
) == pdTRUE )
89:     {
90:       Serial.println(sensorValue);
91:
92:       xSemaphoreGive( xSerialSemaphore );
93:     }
94:
95:     vTaskDelay(1);
96:   }
97: }
```

La comprensión de este código comienza identificando las funciones `setup()` y `loop()` , comenzando en las líneas 11 y 47, típicas de un programa Arduino.

En la función `setup()` , la interfaz en serie se inicializa entre las líneas 14 y 18. Hasta ahora todo bien. Luego viene lo interesante, que es el comienzo de la configuración del sistema multitarea utilizando las funciones de freeRTOS. Yendo un poco más allá (para ignorar la configuración del semáforo durante horas), encontramos la llamada a la función `xTaskCreate()` , entre las líneas 28 y 34:

```
28:   xTaskCreate(
29:     TaskDigitalRead
30:   ,  (const portCHAR *)"DigitalRead"  // A name just for humans
31:   ,  128  // This stack size can be checked & adjusted by reading the Stack Highwater
32:   ,  NULL
33:   ,  2  // Priority, with 3 (configMAX_PRIORITIES - 1) being the highest, and 0 being the lowest.
34:   ,  NULL );
```

Esta llamada es responsable de crear la tarea llamada "DigitalRead" (parámetros 2 en la línea 30), que es un texto asociado a la tarea que se puede usar especialmente para la depuración, un nombre solo humano, como lo llama el autor del código. . El código que se ejecutará es una función típica de C cuya referencia se pasa en el parámetro 1: `TaskDigitalRead()` .

Los otros parámetros de la llamada recibirán nuestra atención más adelante en este libro.

La función de *task* se puede nombrar con cualquier nombre, pero generalmente comienza con Tarea, por convención propuesta en los códigos de muestra y documentación de freeRTOS. Se puede encontrar entre las líneas 56 y 78.

Al comparar esta función con la presentada en el Listado 2 - Pseudocódigo de una tarea junto con una interrupción , notamos la presencia de un comando de inicialización (línea 62: pinMode (`pinMode(pushButton, INPUT);`) y el *loop* infinito de la *task* , entre las líneas 64 y 77.

Este loop fica constantemente lendo o estado de um botão (linha 67) com a função `digitalRead()` do arduino para então enviar pela serial (linha 71). Após isto a *task* espera por algum tempo, chamando a função de delay do freeRTOS chamada `vTaskDelay()` , na linha 76.

Hay dos observaciones muy importantes sobre este retraso (*delay)* :

- **Ya no use** la función arduino `delay()` sino la función freeRTOS `vTaskDelay()` , ya que esta función funciona en armonía con el control de la tarea, aprovechando esta espera para permitir que el código de otra *task* se ejecute nuevamente.
- Después de todo, ¿por qué quedarse con el código en un lugar si tenemos otras tareas que hacer?

- El parámetro de esta función está en *ticks* del *clock* del titular de la tarea, es decir, depende de la configuración de freeRTOS. Por ejemplo, si la *tick* del outras de programación es cada 15 ms, este *delay* es de 15 ms, pero puede cambiar en otras configuraciones. Entonces que hacer?

La solución a esto es simple. FreeRTOS ha proporcionado una macro para convertir el tiempo en microsegundos que queremos, en la cantidad óptima de *ticks* :

```
76:     vTaskDelay( pdMS_TO_TICKS(1000) );
```

También podemos hacer el cálculo descubriendo cuántas millas-segundos por *ticks* tenemos en el sistema, de la siguiente manera:

```
76:     vTaskDelay( 1000 / portTICK_PERIOD_MS );
```

(Ambos ejemplos son para una espera de 1000 ms o 1 segundo)

Esta *task* luego repite el *loop* y vuelve a leer el valor del botón y lo envía a través de la serie, esperando nuevamente antes de ejecutar la próxima vez.

La segunda *task* tiene su código en la función `TaskAnalogRead()` y tiene una estructura similar, pero con un propósito diferente, ya que lee una de las entradas analógicas del arduino y la envía por serie, también de vez en cuando. Se crea en la función `xTaskCreate()` utilizando la función `xTaskCreate()` (línea 36).

Otro detalle, que destaca el autor del código en la línea de comentarios 44, es que después de la configuración, el programador de *task* (*task scheduler*) se inicia automáticamente y las *tasks* creadas comenzarán a ejecutarse a partir de este momento. Esto siempre sucederá a menos que haga algo al respecto, que es suspender la *task* incluso antes de que comience. Este tema se tratará más adelante en este libro.

Dado que ambas tareas tienen prioridades diferentes (cuarto parámetro de la función `xTaskCreate()`), la *task* `TaskDigitalRead()` tendrá mayor prioridad de ejecución que la tarea TaskAnalogRead (), ya que su prioridad es 2 (línea 33), mientras que la otra tiene prioridad 1 (línea 41).

En freeRTOS, un valor numérico más alto corresponde a una prioridad más alta , a diferencia de otros sistemas de prioridad (como las interrupciones de hardware que generalmente tienen la prioridad más alta o el número más bajo).

Consejo : Este es un buen momento para guardar su proyecto y luego ejecutarlo en su arduino. Abra el monitor en serie y verifique los valores de entrada digital y analógica.
La entrada digital contendrá valores entre 0 y 1 (necesitaría conectar la entrada 2, tiempo a 5V, tiempo a GND, para asegurar el valor de su elección). La entrada analógica contendrá valores de 0 a 1023 (de la misma manera que puede montar un circuito con un potenciómetro y conectarse a la entrada analógica A0).
Sin embargo, estas conexiones de hardware no son críticas para comprender este ejemplo.

Pasar parámetros a la *task* en la creación

Podemos tener la misma función de *task* que el código para ejecuciones múltiples o instancias de tareas. Imagine que tiene un solo editor de texto (mismo código) abierto en su PC tres veces, cada uno con un archivo diferente que se está editando. Del mismo modo, puede tener en su sistema integrado (por ejemplo) tres *tasks* con el mismo código pero realizando diferentes operaciones. Para hacer esto posible, la *task* recibe, como parámetro, en el parámetro de función, los datos necesarios para su ejecución diferenciada. Este valor puede ser de un entero o una cadena a una estructura de datos compleja (como se muestra en el ejemplo en el Listado 4).

La definición de la estructura de nuestra *task* , el paso de parámetros y su recepción dentro de la función se resaltarán en la lista y se explicarán más adelante.

Listagem 4: AnalogRead_DigitalReadModificadoVariosLeitores

```
#include <Arduino_FreeRTOS.h>
#include <semphr.h>  // add the FreeRTOS functions for
Semaphores (or Flags).



//tipo de parâmetros

typedef struct {

  const char *pcNomeTask; //nombre de la tarea A CREAR

  const char *pcTexto; //texto a ser enviado por la serie antes
de leer, identificando su fuente

  int SensorID; //ID de entrada analógica utilizada

  int TempoDelayTicks; //tiempo entre enviar lecturas, en ticks
dif. de NULL
```

```
} AnalogReadParametro_t;


// Declare a mutex Semaphore Handle which we will use to manage the Serial Port.

// It will be used to ensure only only one Task is accessing this resource at any time.

SemaphoreHandle_t xSerialSemaphore;


// define two Tasks for DigitalRead & AnalogRead

void TaskDigitalRead( void *pvParameters );

//acepta como parámetro AnalogReadParametro_t

void TaskAnalogReadParam( void *pvParameters );


//establece parámetros para tres tareas de lectura analógica que aceptan parámetros de estructura

AnalogReadParametro_t xParams[3], *pxParam = xParams;



// the setup function runs once when you press reset or power the board

void setup() {


  // initialize serial communication at 9600 bits per second:

  Serial.begin(9600);



  while (!Serial) {
```

```
  ; // wait for serial port to connect. Needed for native USB, on LEONARDO, MICRO, YUN, and other 32u4 based boards.

 }


 // Semaphores are useful to stop a Task proceeding, where it should be paused to wait,

 // because it is sharing a resource, such as the Serial port.

 // Semaphores should only be used whilst the scheduler is running, but we can set it up here.

 if ( xSerialSemaphore == NULL )  // Check to confirm that the Serial Semaphore has not already been created.

 {

  xSerialSemaphore = xSemaphoreCreateMutex();  // Create a mutex semaphore we will use to manage the Serial Port

  if ( ( xSerialSemaphore ) != NULL )

   xSemaphoreGive( ( xSerialSemaphore ) );  // Make the Serial Port available for use, by "Giving" the Semaphore.

 }


 // Now set up two Tasks to run independently.

 xTaskCreate(

  TaskDigitalRead

  ,  (const portCHAR *)"DigitalRead"  // A name just for humans

  ,  128  // This stack size can be checked & adjusted by reading the Stack Highwater

  ,  NULL
```

```
   ,  (configMAX_PRIORITIES - 3)   // Priority, with 3
(configMAX_PRIORITIES - 1) being the highest, and 0 being the
lowest.

   ,  NULL );



   //define parámetros para tres tasks de lectura analógica que
aceptan parámetros de estructura


   pxParam->pcNomeTask = "TaskEntradaA1";

   pxParam->pcTexto = "Entrada A1";

   pxParam->SensorID = A1;

   pxParam->TempoDelayTicks = pdMS_TO_TICKS( 250UL );

   pxParam++;


   pxParam->pcNomeTask = "TaskEntradaA2";

   pxParam->pcTexto = "Entrada A2";

   pxParam->SensorID = A2;

   pxParam->TempoDelayTicks = 1; //pdMS_TO_TICKS( 250UL );

   pxParam++;


   pxParam->pcNomeTask = "TaskEntradaA3";

   pxParam->pcTexto = "Entrada A3";

   pxParam->SensorID = A3;

   pxParam->TempoDelayTicks = 1; //pdMS_TO_TICKS( 250UL );
```

```
   pxParam++;



   //xTaskCreate(TaskAnalogReadParam, pxParam->pcNomeTask, 128,
(void *)&xParams[0], 1, NULL );

   //xTaskCreate(TaskAnalogReadParam, pxParam->pcNomeTask, 128,
(void *)&xParams[1], 1, NULL );

   //xTaskCreate(TaskAnalogReadParam, pxParam->pcNomeTask, 128,
(void *)&xParams[2], 1, NULL );



   //cria as tasks percorre array de ponteiros até o nulo (para
facilitar)

   for(pxParam = &xParams[0]; pxParam < &xParams[3]; pxParam++)

   {

    Serial.println("\n--------------------------------");

    Serial.print("Creando ");

    Serial.println(pxParam->pcNomeTask);



    xTaskCreate(TaskAnalogReadParam, pxParam->pcNomeTask, 128,
(void *)pxParam, (configMAX_PRIORITIES - 2)  /* Priority  */,
NULL );

   }
```

```
// Now the Task scheduler, which takes over control of
scheduling individual Tasks, is automatically started.

}


void loop()

{

  // Empty. Things are done in Tasks.

}


void SerialDebugComSemaforo(const char *pszTexto)

{

   if ( xSemaphoreTake( xSerialSemaphore, ( TickType_t ) 5 ) ==
pdTRUE )

   {

     Serial.print(pszTexto);


     xSemaphoreGive( xSerialSemaphore ); // Now free or "Give"
the Serial Port for others.

   }

}


/*-----------------------------------------------------*/

/*---------------------- Tasks ---------------------*/

/*-----------------------------------------------------*/
```

```
void TaskDigitalRead( void *pvParameters
__attribute__((unused)) )  // This is a Task.

{

  /*

    DigitalReadSerial

    Reads a digital input on pin 2, prints the result to the
serial monitor


    This example code is in the public domain.

  */


  // digital pin 2 has a pushbutton attached to it. Give it a
name:

  uint8_t pushButton = 2;


  // make the pushbutton's pin an input:

  pinMode(pushButton, INPUT);


  for (;;) // A Task shall never return or exit.

  {

    // read the input pin:

    int buttonState = digitalRead(pushButton);


    // See if we can obtain or "Take" the Serial Semaphore.

    // If the semaphore is not available, wait 5 ticks of the
Scheduler to see if it becomes free.
```

```
   if ( xSemaphoreTake( xSerialSemaphore, ( TickType_t ) 5 ) == pdTRUE )
   {
     //Enviar texto literal
     Serial.print("entrada digital: ");

     // We were able to obtain or "Take" the semaphore and can now access the shared resource.
     // We want to have the Serial Port for us alone, as it takes some time to print,
     // so we don't want it getting stolen during the middle of a conversion.
     // print out the state of the button:
     Serial.println(buttonState);

     xSemaphoreGive( xSerialSemaphore ); // Now free or "Give" the Serial Port for others.
   }

   vTaskDelay(1);   // one tick delay (15ms) in between reads for stability
  }
}

void TaskAnalogReadParam( void *pvParameters )  // This is a Task.
```

```
{
  //tomando el parámetro como texto de cadena
  AnalogReadParametro_t *pxParams = (AnalogReadParametro_t *) pvParameters;

  for (;;)
  {
    // read the input on analog pin:
    int sensorValue = analogRead(pxParams->SensorID);

    // See if we can obtain or "Take" the Serial Semaphore.
    // If the semaphore is not available, wait 5 ticks of the Scheduler to see if it becomes free.
    if ( xSemaphoreTake( xSerialSemaphore, ( TickType_t ) 5 ) == pdTRUE )
    {
      //Enviar texto recibido por parámetro
      Serial.print(pxParams->pcTexto);
      Serial.print(": ");
      // We were able to obtain or "Take" the semaphore and can now access the shared resource.
      // We want to have the Serial Port for us alone, as it takes some time to print,
      // so we don't want it getting stolen during the middle of a conversion.
      // print out the value you read:
```

```
    Serial.println(sensorValue);


    xSemaphoreGive( xSerialSemaphore ); // Now free or "Give"
the Serial Port for others.


   }


   vTaskDelay(pxParams->TempoDelayTicks);  // one tick delay
(15ms) in between reads for stability

  }

}
```

Primero definimos el tipo `AnalogReadParametro_t` , que no es más que una estructura con datos de nombre de *task* , el texto que debe enviar la serie antes de leer, el identificador del sensor desde el que debe leer la *task* y el tiempo de *delay* dentro. de cada *task* Podemos agrupar nuestra información en tipos y pasar una variable de este tipo por puntero a la *task* , lo que nos permite configurar el comportamiento de cada *task* específica.

```
//tipo de parámetros

typedef struct {

const char *pcNomeTask; //nombre de la tarea A CREAR

const char *pcTexto; //texto a ser enviado por la serie antes de
leer, identificando su fuente

int SensorID; //ID de entrada analógica utilizada

int TempoDelayTicks; //tiempo entre enviar lecturas, en ticks
caso dif. de NULL

} AnalogReadParametro_t;
```

A continuación se muestra el prototipo de la función de *task* . Las *tasks* siempre deben tener la misma firma, por lo que su parámetro no puede ser del tipo `AnalogReadParametro_t` sino del tipo `void *` . Esto no es realmente un problema, ya que al crear una instancia de la *task* , puede pasar el puntero a la estructura mediante cast para `void *` y leer, dentro de la función, emitir al tipo apropiado (en este caso `AnalogReadParametro *`):

```
//acepta como parámetro AnalogReadParametro_t
void TaskAnalogReadParam( void *pvParameters );
```

Luego declaramos una matriz(*array*) de 3 posiciones del tipo de parámetro, así como un puntero para atravesar esta matriz.

```
//define parámetros para tres tasks de lectura analógica que aceptan parámetros de estructura
AnalogReadParametro_t xParams[3], *pxParam = xParams;
```

Dentro de la función `Setup ()` , cada elemento de esta matriz se inicializará y luego, en un *loop* , las *tasks* se crearán solo con los parámetros apropiados. Debido a que estas son operaciones puramente de lenguaje, iremos directamente a la creación de *tasks* , lo cual es más interesante:

```
xTaskCreate(TaskAnalogReadParam, pxParam->pcNomeTask, 128, (void *)pxParam, (configMAX_PRIORITIES - 2)  /* Priority  */ , NULL );
```

Los datos de parámetros se utilizan para nombrar la *task* en sí pasando `pxParam->pcNomeTask` , mientras que el parámetro entero se pasa por puntero y se convierte en `void *` em `(void *)pxParam` .

Siempre se debe tener en cuenta que la función de *task* siempre es `TaskAnalogReadParam` . Es como si el conjunto entre el nombre de la función `TaskAnalogReadParam()` y los parámetros pasados (`pxParam`) sean una clase que se instancia en tantos objetos como sea necesario.

Finalmente, tenemos la función de *task* , que era una modificación de la función `TaskAnalogRead()` , del listado anterior, pero dependiendo del parámetro para todo (desde decidir el texto a enviar, hasta el origen de la lectura y el tiempo de *delay* entre lecturas). Depende del programador saber

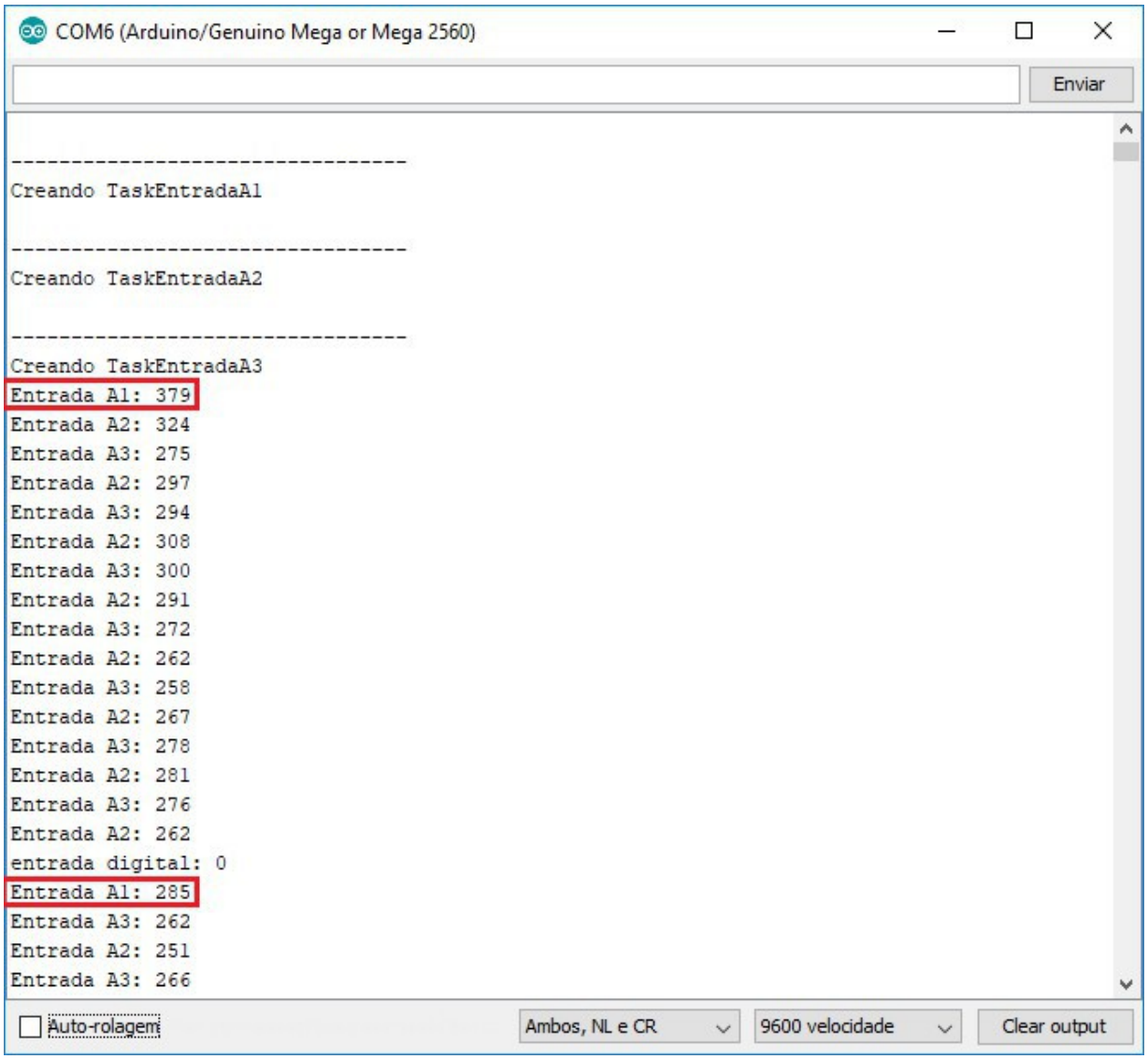
qué es diferente para cada instancia y qué es común a todos. A continuación revisaremos la recepción del parámetro:

```
AnalogReadParametro_t *pxParams = (AnalogReadParametro_t *) pvParameters;
```

Tenemos la declaración de una variable local (`AnalogReadParametro_t *pxParams`), que apunta a la variable pasada por parámetro, con el tipo de conversión apropiado: `(AnalogReadParametro_t *) pvParameters` .

Importante: cada tarea debe tener sus datos para evitar la competencia entre regiones de memoria. Recomendamos que se inicialicen y luego se lean. Si se van a recibir o enviar datos, se deben utilizar mecanismos RTOS como semáforos y colas.

Ejecute este código en arduino y vea que el monitor en serie mostrará lecturas de las tres entradas analógicas A1, A2 y A3:



Puede notar que la *task* que lee la entrada A1 se ejecuta en un intervalo mucho mayor, en la imagen de arriba está rodeada por un rectángulo. Esto se debe a que su parámetro `TempoDelayTicks` se inicializa con la cantidad de *ticks* equivalente a 250 ms, mientras que los otros son solo 1 *tick* (que es equivalente al arduino ATMega2560 a 15 ms). Si solo coincide con intervalos, verá que los tres también envían la misma cantidad de lecturas.

La opción para crear *tasks* en un *loop* era demostrar que puede tener alguna forma de decisión dinámica (no solo fijada en código) leyendo (desde EEPROM, por ejemplo) una configuración y decidiendo cuántas *tasks* leerían los valores y que valores La opción más tradicional fue

comentada y puede reemplazar el *loop* `for ()` al descomentar las siguientes tres líneas (y eliminar o comentar el for):

```
xTaskCreate(TaskAnalogReadParam, pxParam->pcNomeTask, 128,
(void *)&xParams[0], 1, NULL );

   xTaskCreate(TaskAnalogReadParam, pxParam->pcNomeTask, 128,
(void *)&xParams[1], 1, NULL );

  xTaskCreate(TaskAnalogReadParam, pxParam->pcNomeTask, 128,
(void *)&xParams[2], 1, NULL );
```

Detener y reiniciar la ejecución de tareas

Una *task* creada comenzará a ejecutarse automáticamente cuando se inicie la programación de la tarea (que ocurre después de que se termina la función Arduino `Setup()`).

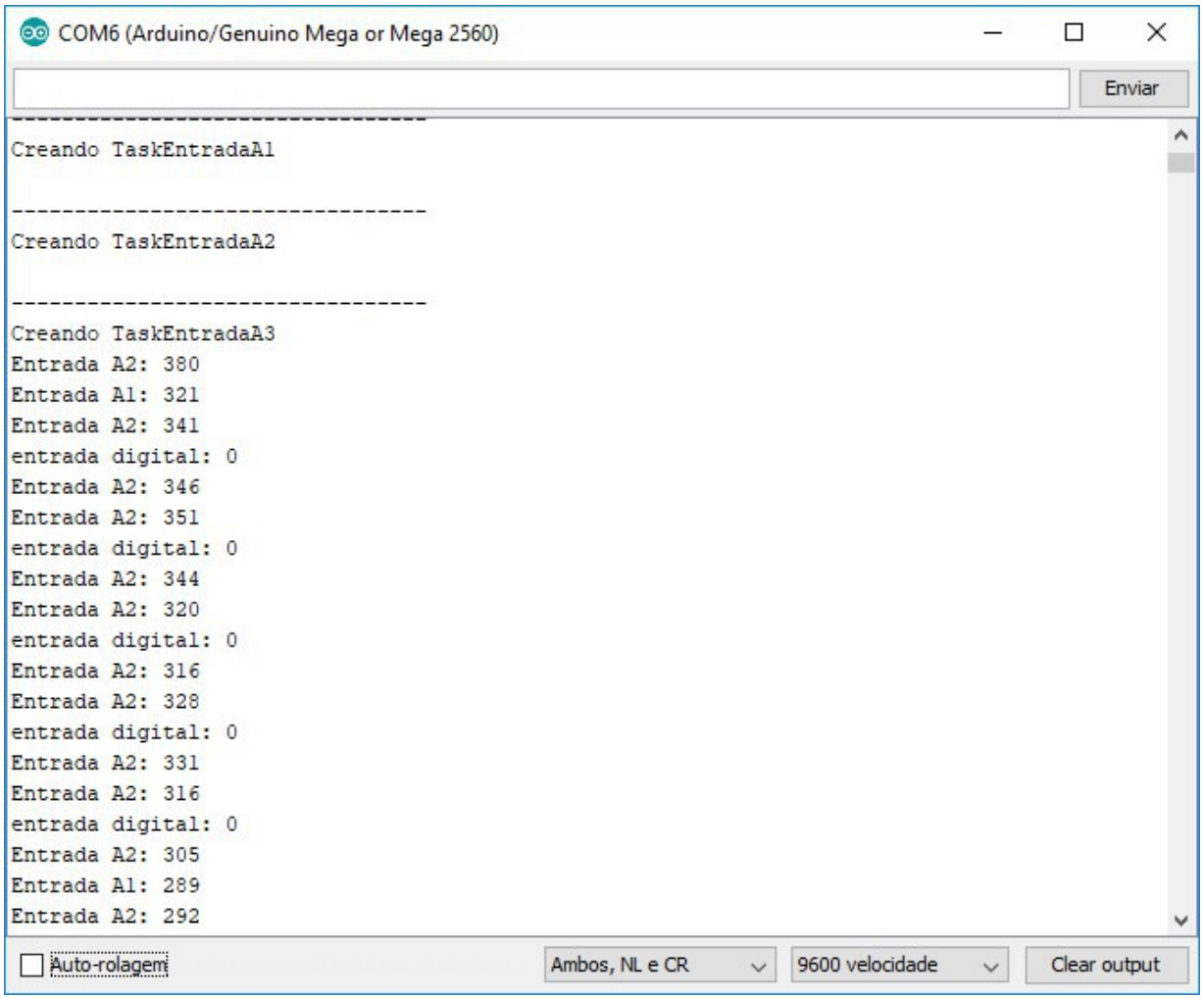
Si queremos que la *task* se suspenda, se detenga sin hacer nada, podemos suspenderla.Si esto se hace antes del final de la función `Setup()` , ni siquiera comenzará la ejecución. Si algún evento o tiempo desencadena esta acción de suspensión, se ejecutará hasta que esto ocurra.

Deberá usar la función `vTaskSuspend (TaskHandle_t pxTaskToSuspend);`

Sin embargo, espera un identificador de *task* , que no tenemos en nuestro ejemplo, ya que es devuelto por la función que crea la *task* en su último parámetro. De esta manera, puede cambiar el código de la lista anterior para guardar la última *task* creada, declarando una variable para eso, guardando el valor de cada *task* en el *loop* y después de que suspenda la última *task* , ya que siempre guardamos la última. El cambio se vería así:

```
       //almacenar la última tarea creada
       TaskHandle_t xUltimaTask;
       //las tasks se ejecutan a través de una variedad de
punteros a nulo (para facilitar)
     for(pxParam = &xParams[0]; pxParam < &xParams[3];
pxParam++)
     {
       Serial.println("\n--------------------------------");
       Serial.print("Creando ");
       Serial.println(pxParam->pcNomeTask);

       xTaskCreate(TaskAnalogReadParam, pxParam->pcNomeTask,
128, (void *)pxParam, (configMAX_PRIORITIES - 2)  /*
Priority  */, &xUltimaTask );
     }

       vTaskSuspend(xUltimaTask);
```

Si ejecuta el programa, verá que la *task* de lectura de A3 nuca realiza:



El pase de la *handle* (indetificador) como parámetro ocurre de manera similar a la estructura:

```
,  (void *)xUltimaTask // Pase el valor hadle de la tarea de conversión a (void *)
```

Ahora, para que la *task* se ejecute nuevamente, presumiremos algún evento al hacerlo llamando a la función `void vTaskResume( TaskHandle_t pxTaskToResume );`.

Para esto, necesitamos que la *task* maneje dónde la vamos a suspender. Una idea es tener esto en el parámetro de las *tasks* y definir para cualquiera de ellos, que será responsable de iniciarlo.

La *task* elegida es la que lee la entrada digital (`TaskDigitalRead()`), para comenzar la *task* cuando su entrada se convierte en 1, que se puede obtener conectando VCC a la entrada digital 2.

Sin embargo, necesitaremos poner su creación después de que se hayan creado las *tasks* que leen las entradas digitales y pasarle el identificador de *task* por parámetro:

```
    .
    .
    .

        xTaskCreate(TaskAnalogReadParam, pxParam->pcNomeTask,
128, (void *)pxParam, (configMAX_PRIORITIES - 2)  /*
Priority  */ , &xUltimaTask );
      }
      vTaskSuspend(xUltimaTask);

       // Now set up two Tasks to run independently.
      xTaskCreate(
        TaskDigitalRead
        ,  (const portCHAR *)"DigitalRead"  // A name just for
humans
        ,  128  // This stack size can be checked & adjusted
by reading the Stack Highwater
        ,  (void *)xUltimaTask // Pase el valor hadle de la
tarea de conversión a (void *)
        ,  (configMAX_PRIORITIES - 3)  // Priority, with 3
(configMAX_PRIORITIES - 1) being the highest, and 0 being the
lowest.
        ,  NULL );
```

A task da entrada digital, com as mudanças necessárias em negrito é apresentada abaixo:

```
    void TaskDigitalRead( void *pvParameters)  // This is a
Task.
   {
      /*
      DigitalReadSerial
```

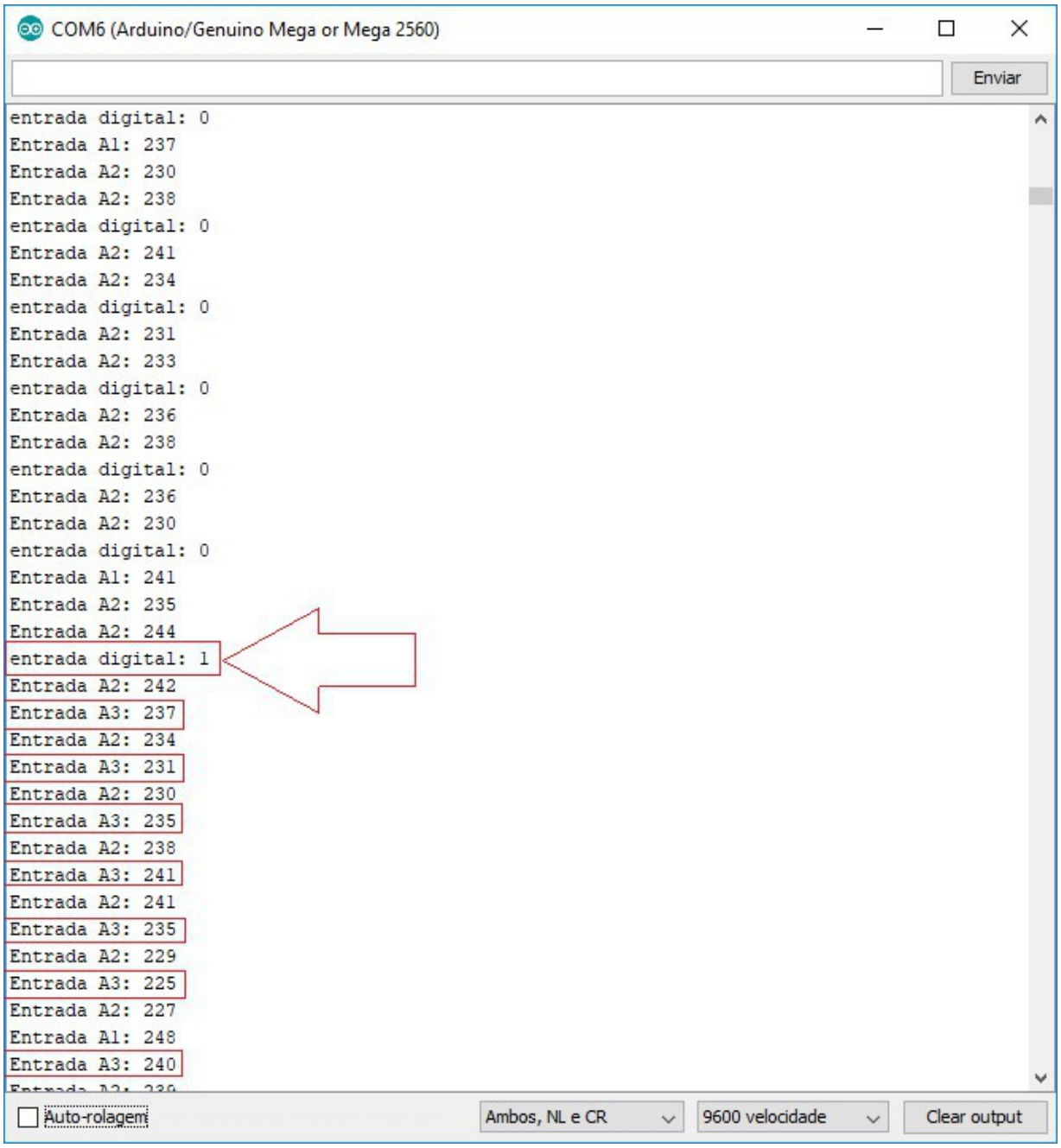
```
      Reads a digital input on pin 2, prints the result to the
serial monitor

      This example code is in the public domain.
     */

      // digital pin 2 has a pushbutton attached to it. Give it
a name:
     uint8_t pushButton = 2;

      TaskHandle_t xTaskParaReiniciar = (TaskHandle_t)
pvParameters;

      // make the pushbutton's pin an input:
     pinMode(pushButton, INPUT);

     for (;;) // A Task shall never return or exit.
     {
       // read the input pin:
      int buttonState = digitalRead(pushButton);

       // See if we can obtain or "Take" the Serial Semaphore.
      // If the semaphore is not available, wait 5 ticks of the
Scheduler to see if it becomes free.
      if ( xSemaphoreTake( xSerialSemaphore, ( TickType_t ) 5 )
== pdTRUE )
      {
        // Enviar texto recibido por parámetro
        Serial.print("entrada digital: ");

        // We were able to obtain or "Take" the semaphore and
can now access the shared resource.
        // We want to have the Serial Port for us alone, as it
takes some time to print,
        // so we don't want it getting stolen during the middle
of a conversion.
        // print out the state of the button:
        Serial.println(buttonState);

        xSemaphoreGive( xSerialSemaphore ); // Now free or
"Give" the Serial Port for others.

        // Decide si reiniciar la tarea
        if( (xTaskParaReiniciar != NULL) && (buttonState == 1)
)
        {
```

```
          vTaskResume(xTaskParaReiniciar);
        }

      }

       vTaskDelay(1);  // one tick delay (15ms) in between
reads for stability
    }
   }
```

Tenga en cuenta que el identificador de *task* (o *hadle*) que se va a reiniciar se lee del parámetro y al cambiar la entrada al nivel lógico 1 se restablece. También se tomó el atributo que decía que el parámetro `pvParameters` no se usó (`__attribute__((unused))`).

Aunque, después de la primera vez que se reinicia la *task* , se ignoran otras llamadas a esta operación cuando la *task* comienza a ejecutarse.

Ejecute el proyecto con estas modificaciones y obtendrá una respuesta de la siguiente manera (la flecha que indica cuándo va la entrada a uno y el otro resalta que muestra que la *task* se ha vuelto a ejecutar):



Este código completo se puede encontrar en el Apéndice A.1 - AnalogRead_DigitalREadModifiedVariousReaders (con suspensión de tasks) . Úselo para comparar sus modificaciones y encontrar cualquier error de codificación o simplemente para ver más de cerca el ejemplo completo.

Capítulo 4 - Trabajando con semáforos

Trabajemos con ejemplos

En el Listado 3: AnalogRead DigitalRead (ejemplo de biblioteca sin algunos comentarios) ya vimos un semáforo MUTEX en acción, aunque en ese momento el foco no estaba en el control de acceso en serie.

El siguiente es un ejemplo más simple, que envía dos textos pasados por parámetro a cada una de las *tasks* que tienen la misma función. De esta forma, solo el parámetro diferencia la ejecución de una *task* de otra. Ejecutaremos este proyecto y entenderemos cómo actúa el semáforo. Luego apagaremos el semáforo y veremos el resultado en el monitor de comunicación en serie.

Semáforo MUTEX

Examine el código en nuestro ejemplo, que se explicará a continuación.

GitHub

Listagem 5:
ExemploSemaforoMutexSerial

```
01: #include <Arduino_FreeRTOS.h>
02: #include <semphr.h>
03:
04: SemaphoreHandle_t xSerialSemaphore;
05:
06: void TaskEnviaTextoSerial( void *pvParameters );
07:
08: void setup()
09: {
10:    Serial.begin(9600);
11:    while (!Serial) ;
12:
13:
14:    if ( xSerialSemaphore == NULL )  // Check to confirm that
the Serial Semaphore has not already been created.
15:    {
16:      xSerialSemaphore = xSemaphoreCreateMutex();  // Create a
mutex semaphore we will use to manage the Serial Port
17:      if ( ( xSerialSemaphore ) != NULL )
```

```
18:        xSemaphoreGive( ( xSerialSemaphore ) );  // Make the
Serial Port available for use, by "Giving" the Semaphore.
19:    }
20:
21:    if ( xSerialSemaphore != NULL )
22:    {
23:      // Cree dos tareas con la misma función, pasando texto
diferente por parámetro
24:      xTaskCreate(TaskEnviaTextoSerial, "TaskEnviaTextoSerial1",
128, (void *)"Texto 1", 1, NULL );
25:      xTaskCreate(TaskEnviaTextoSerial, "TaskEnviaTextoSerial2",
128, (void *)"Texto 2", 1, NULL );
26:    }
27:    else
28:    {
29:      Serial.println("**** Error al crear semáforo!");
30:    }
31: }
32:
33: void loop()
34: {
35:   // Nada en el bucle. ¡Todo en las tareas!
36: }
37:
38: void TaskEnviaTextoSerial( void *pvParameters )
39: {
40:    // tomando el parámetro como texto de cadena
41:   const char *pszTexto = (const char *) pvParameters;
42:
43:   for (;;)
44:   {
45:      // Cambia portMAX_DELAY por (TickType_t) 5 por ticks y
luego abandona
46:     if ( xSemaphoreTake( xSerialSemaphore, portMAX_DELAY ) ==
pdTRUE )
47:     {
48:       Serial.println("-------------------------");
49:       Serial.println(pszTexto);
50:
51:       xSemaphoreGive( xSerialSemaphore );
52:
53:     }
```

```
54:       vTaskDelay(pdMS_TO_TICKS(1000));
55:     }
56: }
57:
58:
```

Vamos a explicar: en la línea 2, tenemos la inclusión del archivo `semphr.h` , que define el tipo y las funciones de los semáforos.

En la línea 4, en el alcance global del programa, tenemos la declaración de semáforo `xSerialSemaphore` , de tipo `SemaphoreHandle_t` . Este tipo es siempre el mismo independientemente del tipo de semáforo que realmente se creará.

A continuación, en la función setup (), verificamos el valor de la variable `xSerialSemaphore` en la línea 14, que en realidad es innecesario, ya que sabemos que no se creó fuera de setup (). Entonces la función responsable de crear un semáforo de tipo mutex se llama: `xSemaphoreCreateMutex()` . Si devuelve un valor distinto de nulo en `xSerialSemaphore` , significa que se creó con éxito. (generalmente debido a la falta de memoria disponible, que claramente no es nuestro caso en este contexto simple).

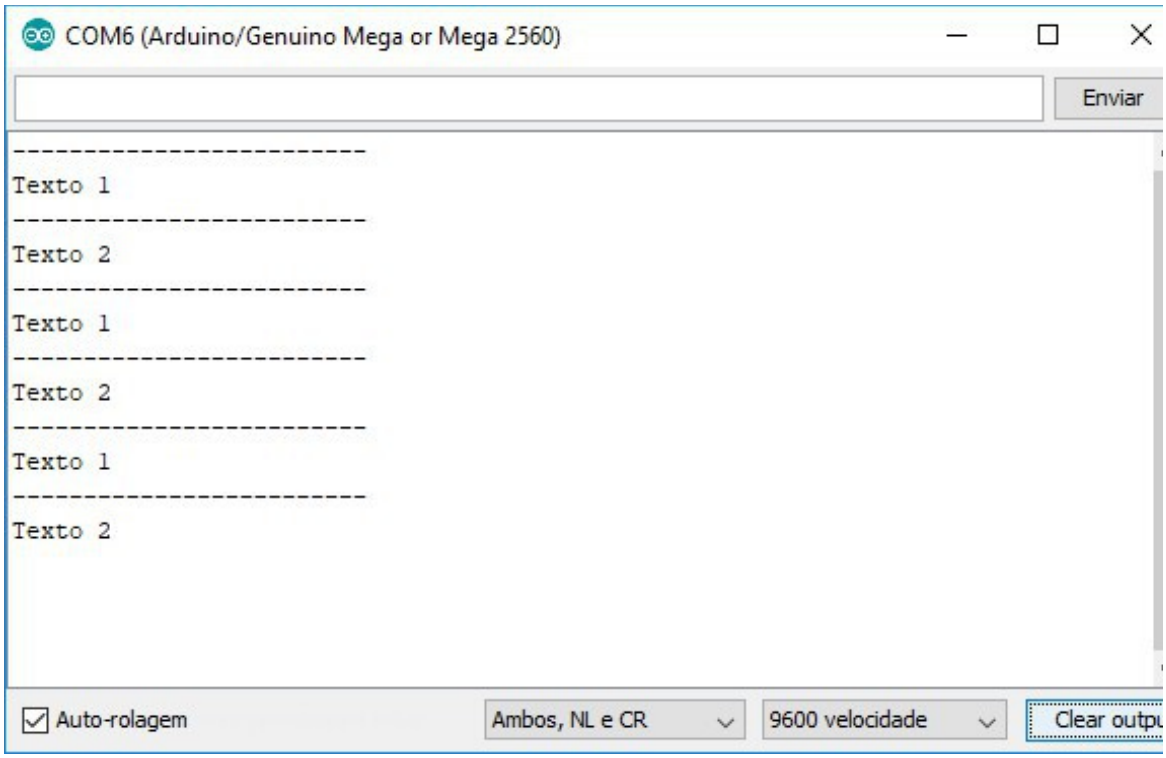
La creación correcta del semáforo se prueba en la línea 17, para luego llamar a la función que libera el semáforo:

```
16:   xSerialSemaphore = xSemaphoreCreateMutex();  // Create a
mutex semaphore we will use to manage the Serial Port
17:   if ( ( xSerialSemaphore ) != NULL )
18:     xSemaphoreGive( ( xSerialSemaphore ) );  // Make the Serial
Port available for use, by "Giving" the Semaphore.
```

Llamar a `xSemaphoreGive` hace que el semáforo se libere y puede ser recogido (asignado) por la primera tarea que lo solicita. Actualmente no tenemos tareas creadas todavía. Esto se hará en las líneas 24 y 25 que crean dos tareas con la misma función `TaskEnviaTextoSerial()` , pero pasando diferentes parámetros de texto:

```
(void *)"Texto 1"
(void *)"Texto 2"
```

Después de la configuración (), ambas tareas comienzan a ejecutarse y podemos ver el siguiente resultado en el monitor en serie:



A medida que se libera el semáforo, la primera tarea que lo solicita, con la función `xSemaphoreTake` , tendrá éxito, como se ve en la línea 46:

Una vez que se asigna el semáforo y, por lo tanto, el acceso al objeto Serie, se envía una secuencia de 25 trazos seguida de un *caracter* de salto de línea y luego el texto recibido por el parámetro mismo.

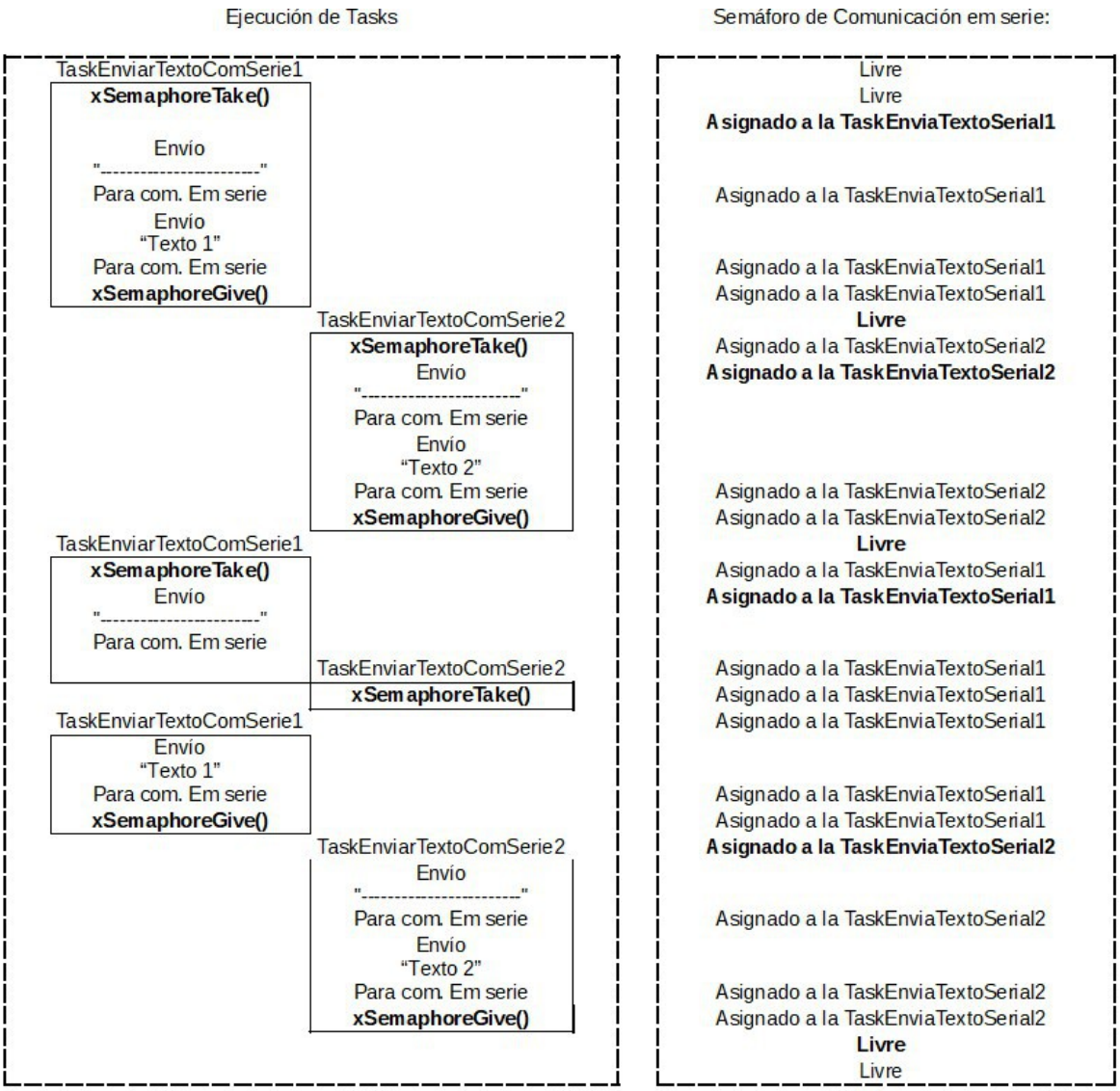
Cuando se completan las dos llamadas a `Serial.println()` , el semáforo se libera con la llamada a la función `xSemaphoreGive` , que libera el semáforo, en la línea 51. De esta manera, la otra tarea que estaba "bloqueada en la llamada a `xSemaphoreTake` , finalmente obtendrá el semáforo. y puede tener el objeto Serie solo para ella, luego suéltelo.

Ele fica esperando indefinidamente por que o segundo parâmetros da função `xSemaphoreTake` é o tempo de espera em ticks (o primeiro é a variável do semáforo, que é global). Se passarmos a constante `portMAX_DELAY` em vez de um valor em *ticks* , como por exemplo 5, ele ficará indefinidamente naquela linha, até que outra *tasks* que estiver travando o semáforo o libere.

Espera indefinidamente porque el segundo parámetro de la función `xSemaphoreTake` es el tiempo de espera de tick (el primero es la variable de semáforo, que es global). Si pasamos la constante `portMAX_DELAY` en lugar de

un valor en ticks, como 5, permanecerá en esa línea indefinidamente hasta que otra *tasks* que bloquee el semáforo la libere.

Esta secuencia se puede ver en la siguiente figura:

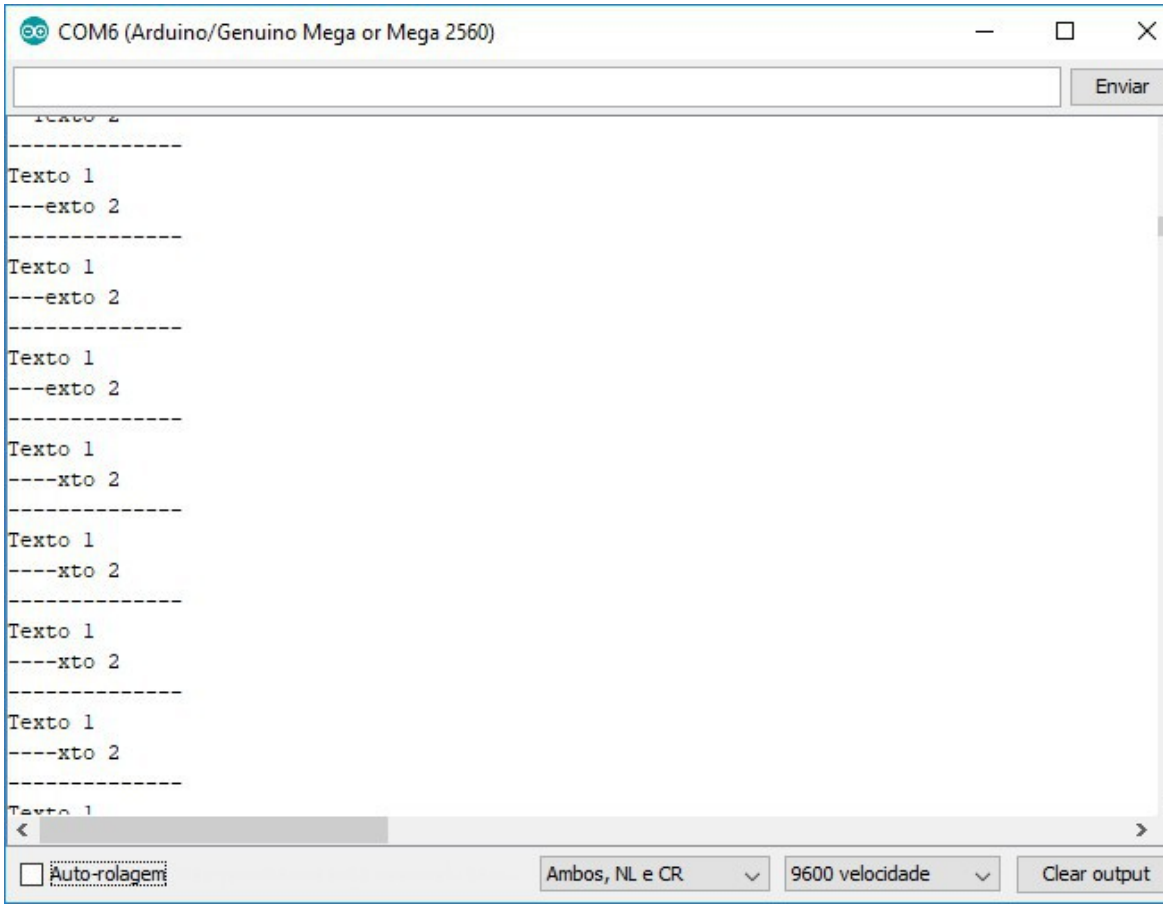


Ahora descartemos el uso del semáforo sin controlar las llamadas simultáneas al método `println()` del objeto Serial y veamos qué sucede. Simplemente comente las líneas 46, 51 y 54. La línea de comentarios 54, que elimina la llamada a `vTaskDelay` , pretende intensificar la disputa por el número de serie y aumentar las posibilidades de problemas. El resultado debe ser similar al siguiente:

```
43:    for (;;)
44:    {
```

```
45:       // Cambia portMAX_DELAY por (TickType_t) 5 por ticks y
luego abandona
46:       //if ( xSemaphoreTake( xSerialSemaphore, portMAX_DELAY ) ==
pdTRUE )
47:       {
48:         Serial.println("-------------------------");
49:         Serial.println(pszTexto);
50:
51:          //xSemaphoreGive( xSerialSemaphore );
52:
53:       }
54:       //vTaskDelay(pdMS_TO_TICKS(1000));
55:     }
```

Ejecute su proyecto y verá en el monitor en serie que el envío de caracteres está codificado, con parte del tiempo del texto enviado por la primera prueba, hora de la segunda:



Este semáforo no solo controla el acceso de dos *tasks* , sino varias. Intente descomentar el control del semáforo y agregue tres *tasks* más, agregue las nuevas llamadas de compilación después de la línea 25, simplemente cambie el texto enviado a la *tasks* y su nombre:

```
    xTaskCreate(TaskEnviaTextoSerial, "TaskEnviaTextoSerial3",
128, (void *)"Texto 3", 1, NULL );
    xTaskCreate(TaskEnviaTextoSerial, "TaskEnviaTextoSerial4",
128, (void *)"Texto 4", 1, NULL );
    xTaskCreate(TaskEnviaTextoSerial, "TaskEnviaTextoSerial5",
128, (void *)"Texto 5", 1, NULL );
```

Deberías obtener el siguiente resultado:

```
-------------------------
Texto 1
-------------------------
Texto 2
-------------------------
Texto 3
-------------------------
Texto 4
-------------------------
Texto 5
-------------------------
Texto 1
-------------------------
Texto 2
-------------------------
Texto 3
-------------------------
Texto 4
-------------------------
Texto 5
-------------------------
Texto 1
```

Semáforo binario

A diferencia de un semáforo MUTEX, que protege a un recurso del acceso por más de un punto de código, causando una exclusión mutua, el semáforo binario actúa como una *flag* . Por supuesto, este indicador no es un simple *flag* , sino que tiene todas las características de un sistema operativo en tiempo real y la programación de *tasks* . Funciona con 1 bit, como su nombre lo indica, pero este bit se puede establecer desde otra tarea o desde una interrupción y luego probar (o esperar) en una *task* .

Para comprender mejor esto, haremos una comparación entre la solución de bit de *flags* y el semáforo binario en un sistema RTOS.

Prueba de bits para manejar eventos

Sin usar un sistema operativo, programar el *firmware* directamente en el hardware, a menudo usamos *flags* de bits para advertir al *loop* en la función `main()` (o alguna función llamada) que se ha producido un evento, generalmente debido a una interrupción. Esto también es cierto para los programas arduino, que utilizan interrupciones en algunos casos para un mayor control del hardware.

Veamos un ejemplo en el que desea monitorear un pin conectado a una tecla y advertir a la computadora (a través de serie) que cada cambio en el pin se realiza en uno, lo que indica que la llave se ha activado.

Listado 6: InterrupcaoPinoChaveFlagParaLoop (sin RTOS)

```
01: const byte PinKey = 2;
02: volatile bool flag_llaveEncendida = false;
03:
04: void setup() {
05:    Serial.begin(9600);
06:
07:    pinMode(PinKey, INPUT_PULLUP);
08:    attachInterrupt(digitalPinToInterrupt(PinKey),
09:      tratarInterrupcionClave, RISING);
10: }
```

```
11:
12: void loop() {
13:    if(flag_llaveEncendida)
14:    {
15:      flag_llaveEncendida = false;
16:      Serial.print("llave encendida");
17:    }
18:
19:     // Aquí se realizarían actividades diversas del sistema,
por ejemplo
20: }
21:
22: void tratarInterrupcionClave() {
23:    fflag_llaveEncendida = true;
24: }
```

En este ejemplo, tenemos una bandera marcada como `volatile` llamada `flag_llaveEncendida` , que se establece en verdadero en la función que activa la interrupción de ascenso (del nivel lógico 0 al 1) desde el pin 2, donde está conectada nuestra clave.

Luego, en la función `loop()` , el indicador se prueba constantemente y, cuando es verdadero(`true`), se establece inmediatamente en cero (`false`) y el texto enviado por el serial. Una vez establecido en `false` , tenemos la opción de detectar un evento próximo en el manejo de la interrupción `tratarInterrupcionClave()` a medida que ocurre.

El comentario "`// Aquí se realizarían actividades diversas del sistema, por ejemplo` " Simula el punto en el que se realizan otras operaciones del sistema, independientemente del control de teclas y el envío a la PC.

Usando un semáforo binario para manejar eventos

No nosso RTOS utilizamos semáforos binários para este fim, de forma que possamos ter todos os benefícios de se poder associar uma *task* para tratar a chave (ou conjunto de chaves), de forma mais flexível e sem precisar ficar enchendo seu programa de *flags* nas interrupções e testes no

loop do programa. Pode-se então ficar com a *task* apostos a espera do semáforo sinalizar que o evento foi indicado, tratando então a informação. Caso a *task* seja a de maior prioridade, ela será imediatamente executada, mesmo que uma outra *task* estivesse em execução antes da interrupção, ganhando muito tempo tratando o evento de forma mais adequada.

Veremos a seguir então a versão o uso de um semáforo binário e as considerações de seu uso e diferença em relação ao semáforo MUTEX.

Listagem 7: InterrupcaoPinoChaveParaTask (usando RTOS)

```
001: #include <Arduino_FreeRTOS.h>
002: #include <semphr.h>
003:
004: const byte PinKey = 2;
005: /////////////// Flag eliminada !! ///////////////
006: //volatile bool flag_llaveEncendida = false;
007:
008:
009: // Semáforo binario para indicar evento clave
010: SemaphoreHandle_t xSemaforoBinarioClave;
011:
012: // Función de task para notificar la vinculación de teclas
013: void TaskClave( void *pvParametros );
014: // Simula otras cosas que el programa necesita hacer
constantemente
015: void TaskOtrasActividadesDelSistema( void *pvParametros );
016:
017:
018: void setup() {
019:
020:   BaseType_t xRet;
021:
022:   Serial.begin(9600);
023:
024:   pinMode(PinoChave, INPUT_PULLUP);
025:
026:    // Crear semáforo binario
```

```
027:    xSemaforoBinarioClave = xSemaphoreCreateBinary();
028:
029:    // Si todo está bien, crea la task
030:    if( xSemaforoBinarioClave != NULL) {
031:
032:       xTaskCreate( TaskOtrasActividadesDelSistema,
033:          ( const portCHAR *) "OtrasAtivSis", 128, NULL, 1 ,
NULL);
034:
035:       xRet = xTaskCreate(
036:         TaskClave
037:        ,  (const portCHAR *) "TaskClave"
038:        ,  128   // Stack size
039:        ,  NULL
040:        ,   2  // Priority
041:        ,  NULL );
042:
043:        if(xRet == pdPASS)
044:        {
045:          attachInterrupt(digitalPinToInterrupt(PinKey),
046:            tratarInterrupcionClave, RISING);
047:        }
048:        else
049:        {
050:          Serial.println("*** Error al crear task");
051:        }
052:    }
053:    else
054:    {
055:      Serial.println("*** Error al crear semáforo");
056:    }
057:
058: }
059:
060: void loop() {
061:    // Indicador de monitoreo dejado aquí
062: }
063:
064: void tratarInterrupcionClave() {
```

```
065:    // Variable para controlar el cambio de task después de
dejar la interrupción.
066:    // Este esquema siempre debe hacerse para usar las
funciones que terminal en FromISR
067:    BaseType_t xHigherPriorityTaskWoken;
068:
069:    // Use el semáforo para indicar el evento clave.
Desbloqueando la task que lo esperaba.
070:    // Pase la dirección xHigherPriorityTaskWoken para
tratar después del planificador antes de salir
071:    // de la interrupción.
072:
073:    xSemaphoreGiveFromISR( xSemaforoBinarioClave,
&xHigherPriorityTaskWoken );
074:
075:    // Esta es la llamada que debe hacerse al final de ISR
() para que freeRTOS cambie las tasks,
076:    //en su caso
077:    if( xHigherPriorityTaskWoken )
078:    {
079:      taskYIELD ();
080:    }
081: }
082:
083: // Función de task para notificar la vinculación de teclas
084: void TaskClave( void *pvParametros ) {
085:    // loop infinito de tarea
086:    for( ;; )
087:    {
088:      // Espera la señal de que la llave está encendida
089:      xSemaphoreTake( xSemaforoBinarioClave, portMAX_DELAY );

090:      // Enviar información en serie
091:      Serial.print("llave encendida");
092:    }
093:
094: }
095:
```

```
096: // Simula otras cosas que el programa necesita hacer
constantemente
097: void TaskOtrasActividadesDelSistema( void *pvParametros ) {
098:
099:    // loop infinito
100:    for(;;) {
101:      // Aquí se realizarían actividades diversas del
sistema, por ejemplo
102:    }
103: }
```

En la línea 6, se comenta la declaración de `flag_llaveEncendida` para mostrar que este tipo de mecanismo ya no se usa, sino el semáforo, cuya variable global se define en la línea 10:

```
010: SemaphoreHandle_t xSemaforoBinarioClave;
```

Nota: Puede observar que el tipo de la variable es el mismo que el semáforo MUTEX, lo que hace que las cosas sean más flexibles, por lo que puede pasar un semáforo a una función que lo maneja, sin saber exactamente cuál es su tipo real (en orientación al objeto esto se asocia con polimorfismo). La función de creación que inicializa los datos dentro de la estructura de tipos de manera apropiada para cada tipo mientras se mantiene el tipo `SemaphoreHandle` . Imagine que desea ocultar un poco los detalles o crear una función de nombre más descripble en espanol. Puede tener la siguiente firma:

```
ObtenerSemaforo(SemaphoreHandle_t xSemaforoBinarioChave);
```

ou então:

```
DispararSemaforo(SemaphoreHandle_t xSemaforoBinarioChave);
```

Para hablar sobre dos posibilidades ... La función realmente no necesita saber qué tipo de semáforo, ya que esta operación se puede realizar en todos los tipos de semáforos con diferentes significados, pero llamando a la misma función que la API freeRTOS: `xSemaphoreTake(` .

A continuación tenemos dos funciones de tareas: `TaskClave()` y `TaskOtrasActividadesDelSistema()` en las líneas 13 y 15, la primera es la

tarea dedicada a monitorear los cambios clave a través del semáforo binario.

La segunda *task* tiene la función de ejemplificar las otras operaciones del sistema, que varían según su propósito. Reemplazan la misma simulación que en la lista anterior en la función de bucle, pero ahora con una *task* .

La función `setup()` se ha cambiado para crear el semáforo binario después de la configuración del pin en la línea 27:

```
027:    xSemaforoBinarioClave = xSemaphoreCreateBinary();
```

Tenga en cuenta que se utiliza la función `xSemaphoreCreateBinary()` en lugar de la función `xSemaphoreCreateMutex ()` para que se cree el tipo de semáforo binario. En la línea 30 se prueba la variable. Si nulo significa que se ha producido un error (que se informa en las líneas "else" 53 a 56, enviando un texto a través de la serie). El error previsto está básicamente fuera de memoria, en la asignación dinámica de memoria (que no se espera en la asignación temprana de software, a menos que ya haya asignado toda la memoria en alguna matriz gigante declarada estáticamente, por ejemplo).

Entonces, con el semáforo creado con éxito, las *tasks* se crearán en las líneas 32 y 35. La retroalimentación de éxito se ha implementado solo para la segunda *task* , pero la solución más sólida es probar la creación de retorno de ambas tareas. Esta verificación se realiza para la segunda *task* (que trata con la clave) en las líneas 48 a 51. Tenga en cuenta que las pruebas de retorno consisten en comparar una variable de tipo `BaseType_t` con `pdPASS . PdPASS` es una constante utilizada para devolver esta función ejecutada con éxito.

Consejo: cada adaptación de freeRTOS para una plataforma específica define `BaseType_t` como el tipo que coincide con el tamaño de palabra en esa plataforma. Entonces, si su plataforma es de 8 bits, este tipo será de 8 bits. Lo mismo ocurre con 16, 32 y 64 bits.

El uso de este tipo hace que las operaciones se realicen con el tipo que será más rápido. Deberíamos usar estos tipos para prevenir problemas de diferentes tipos.

Luego tenemos, en la línea 64, la función `tratarInterrupcionClave()` que trata la interrupción externa en el pin de la llave, como antes, pero ahora adaptada para usar el semáforo creado.

Aquí hay un punto crucial: al verificar la clave y enviar una señal con el semáforo que se ha activado, esperamos que las *tasks* interesadas en esta información sean notificadas lo antes posible. Después de que se haya notificado el evento (con el semáforo), la *task* que lo espera podrá ejecutarse nuevamente y puede ejecutarse después de la interrupción, si tiene más prioridad que la que se está ejecutando, como es nuestro caso.

Podemos ver el evento marcado con la llamada de función `xSemaphoreGiveFromISR` en la línea 73. Tenga en cuenta que la función realizada es especial para interrupciones (termina en `FromISR`). La variable `xHigherPriorityTaskWoken` viene con información freeRTOS sobre si se deben cambiar las tareas. El sistema freeRTOS necesita un poco de ayuda dentro de la interrupción para cambiar las tareas. Se usa en la línea 77, llamando a la función (o macro C) `taskYIELD()` si la variable es verdadera. Esta función, como dije, cambia los controles de ejecución para realizar la siguiente tarea justo antes de abandonar la interrupción.

En la línea 84 finalmente tenemos la *task* que espera la señal señalada en la interrupción (`xSemaforoBinarioClave`). La función espera indefinidamente el semáforo (ya que pasa el parámetro `portMAX_DELAY` como tiempo de espera) en la línea 89, ya que la función xSemaphoreTake () bloquea la ejecución de esta *task* `xSemaphoreTake` hasta el semáforo. Tenga en cuenta que la *task* ni siquiera se ejecuta de vez en cuando, porque freeRTOS, sabiendo que espera el semáforo, la elimina temporalmente de la lista de *tasks* que puede realizar de vez en cuando.

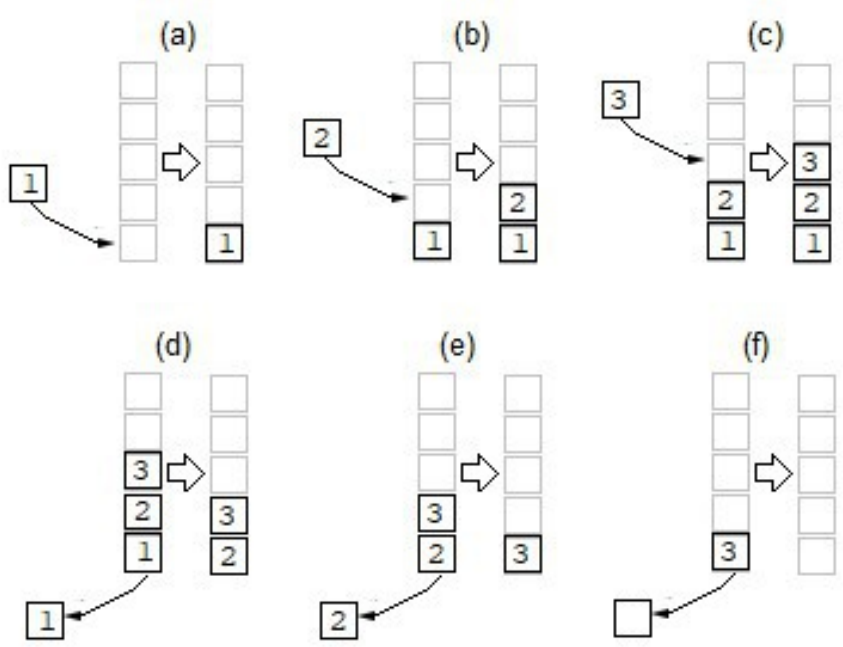
Hemos agregado la tarea `TaskOtrasActividadesDelSistema()` (línea 97) para mostrar las otras cosas que el sistema estaría haciendo mientras la tecla no se activa. Es por eso que en un programa real tendrías esta y otras tareas, cada una encargada de sus propios asuntos. Puede imaginar esta tecla como la tecla de encendido de su *smartphone* , que responde con un menú que le permite, entre otras cosas, apagar su teléfono.

Capítulo 5 - Colas de datos

Comunicación entre tasks

Cuando necesitamos comunicar varios elementos entre partes de nuestro código, generalmente pasamos matrices (*arrays*) por parámetros o usamos variables globales y accedemos entre los dos puntos.

Otro mecanismo que se puede utilizar es una cola, con una o más posiciones que se comunican sincrónicamente entre *tasks* o interrupciones y *tasks* . A medida que veamos mejor cómo se puede hacer esto, imaginemos que tenemos una fila de números y que una *task* es poner los números en secuencia. Primero ponga el número 1, luego el 2 y finalmente el número 3. La segunda *task* está consumiendo estos elementos, es decir, de vez en cuando recoge los elementos de la cola, si los hay. En aras de la simplicidad, consideremos que la segunda *task* solo comienza a eliminar elementos después de que la primera haya ingresado los tres [2] .



Nota: Cada paso indica el paso anterior y posterior, separador por una flecha de izquierda a derecha.

Pasos:

(a) task1 agrega el elemento 1 a la cola vacía;

(b) la task1 se agrega a la cola que ya tiene el ítem 1, ítem 2;

(c) la task1 se agrega a la cola que ya tiene los elementos 1 y 2, el elemento 3;

(d) La task2 elimina el elemento 1 de la cola y lo deja con dos elementos (valores 2 y 3);

(e) La task2 elimina el elemento 2 de la cola y lo deja con el elemento 3;

(f) La task2 elimina el elemento 3 de la cola dejándolo vacío.

Ahora para un ejemplo práctico!

Nuestro primer ejemplo de código en cola consistirá en una función de interrupción y una *task* , la primera que monitorea una clave (basada en Listagem 7: InterrupcaoPinoChaveParaTask (usando RTOS)) y cada disparador enviará un mensaje de texto a la *task* ser enviado a serie (y visto en el monitor serie Arduino).

Antes de introducir el código, algunos comentarios sobre nuestra cola. Tendrá una cantidad de elementos (diez), cada elemento tendrá un tamaño de 21 bytes, ya que podemos poner el tipo de datos que queremos en la cola (en el ejemplo de la figura era un valor numérico y aquí una cadena). Este tamaño de 21 bytes es suficiente para colocar texto de hasta veinte caracteres (ya que un byte debe reservarse para el final del *caracter* nulo). Entonces puedes poner textos como el siguiente:

```
El primer disparador
Clave activada 2
Tecla activada 3
Tecla activada 4
Tecla activada 5
```

Segue ilustração de como ficariam estes itens na fila:

Para tomar este ejemplo de inmediato, crearemos un nuevo programa similar al Listagem 7: InterrupcaoPinoChaveParaTask (usando RTOS) , pero usando la cola en lugar del semáforo. De esta manera solo podemos resaltar la parte de la línea, ya que el resto ya se ha entendido. Sigue el código:

GitHub

Listagem 8:
FilaTextoEnvioChave

```
001: #include <Arduino_FreeRTOS.h>
002: #include <queue.h>
004: const byte PinoChave = 2;
005:
001: #include <Arduino_FreeRTOS.h>
002: #include <queue.h>
004: const byte PinKey = 2;
005:
006: //Identificador de cola
007: QueueHandle_t xCola;
008:
009: char szTextoPrimerElemento[] = "El primer disparador";
010: char  szTextoOtrosElementos[] = "Clave activada X";
011: char nContador = 0;
012:
```

```
013: //Función de task para enviar por comunicación serial
014: void TaskEnviarComunicacionSerial( void *pvParametros );
015:
016:
017: void setup() {
018:
019:   BaseType_t xRet;
020:
021:   Serial.begin(9600);
022:   attachInterrupt(digitalPinToInterrupt(PinKey),
tratarInterrupcionClave, RISING);
023:   pinMode(PinKey, INPUT_PULLUP);
024:
025:    // Crea la cola de carga de personajes
026:   xCola = xQueueCreate(
027:     10,                  // Longitud de la cola: 10
028:     21 *sizeof( char )   // Tamaño del elemento: 21
029:   );
030:
031:
032:   if(xCola) Serial.println("Cola creada exitosamente");
033:
034:   xRet = xTaskCreate(TaskEnviarComunicacionSerial, (const
portCHAR *) "TaskEnviarComSer",  128, NULL, 2, NULL);
035:
036:   if(xRet == pdPASS) Serial.println("Task creada con éxito:");
037: }
038:
039: void loop() {
040:   // Indicador de monitoreo dejado aquí
041: }
042:
043:
044: void tratarInterrupcionClave() {
045:   BaseType_t xHigherPriorityTaskWoken;
046:
047:   nContador++;
048:   if(nContador > 9) nContador = 2;
049:
050:   if(nContador == 1)
051:   {
052:     // pone en cola el elemento pasando el texto específico
al primer elemento
053:     xQueueSendToBackFromISR( xCola, (void
*)szTextoPrimerElemento, &xHigherPriorityTaskWoken);
054:   }
055:
056:   else
```

```
057:   {
058:     // Ensambla el texto para copiar en el elemento de la
cola
059:      szTextoOtrosElementos[15] = '0' + nContador;
060:
061:     // poner el elemento en la cola pasando el texto a copiar

062:     xQueueSendToBackFromISR( xCola, & szTextoOtrosElementos,
&xHigherPriorityTaskWoken);
063:   }
064:
065:   if( xHigherPriorityTaskWoken )
066:   {
067:     taskYIELD ();
068:   }
069: }
070:
071: // Función de task para notificar la vinculación de teclas
072: void TaskEnviarComunicacionSerial( void *pvParametros ) {
073:    // Texto almacenado para ser recibido
074:   char szTexto[21];
075:
076:   Serial.println("loop inicio: Clave de tareas");
077:
078:    // loop infinito de tarea
079:   for( ;; )
080:   {
081:     // Espera indefinidamente hasta leer un elemento de la
cola
082:     xQueueReceive( xCola, &szTexto, portMAX_DELAY );
083:
084:     Serial.print(szTexto);
085:   }
086: }
```

La variable de cola se declara globalmente en la línea 7 con el tipo `QueueHandle_t` . La creación de la cola ya se realiza en las líneas 26 a 29:

```
026:   xCola = xQueueCreate(
027:     10,                  // Longitud de la cola: 10
028:     21 *sizeof( char )   // Tamaño del elemento: 21
029:   );
```

En la línea 26 se llama a la función `xQueueCreate()` con dos parámetros para crear la cola, y en el primer parámetro pasamos el tamaño de la cola (en

este caso, 10 elementos) y en el segundo parámetro pasamos el tamaño, en bytes, de cada elemento. Usamos el operador `sizeof` para obtener el número de bytes de tipo `char` y luego multiplicamos por 21.

La variable `xCola` recibe la referencia a la cola que se utilizará tanto en la interrupción que envía datos a la cola como en la *task* que lee estos datos. En la línea 32, esta variable se prueba, y si tiene un valor verdadero se señaliza en serie [3] .

Ahora usemos la cola:

Entre las líneas 44 y 69 tenemos la función `tratarInterrupcionClave()`, que implementa el servicio externo de manejo de interrupciones asociado con nuestro pin clave. En nuestro caso, esta función se llama cada vez que se presiona una tecla para poner en cola un elemento de texto.

Nota: Tenga en cuenta que manipular texto no es el mejor enfoque en una interrupción, ya que debe ser lo más breve posible, pero servirá como primer ejemplo, que luego se modificará también en este punto.

¡Pues bien! Inicializamos tres variables globales que solo son utilizadas por esta interrupción (que elimina la competencia de datos): `szTextoPrimerElemento`, `szTextoOtrosElementos` y `nContador`. Los dos primeros se utilizan para determinar el texto que se enviará (el primero fijo en la línea 53 y el segundo editado antes del envío en la línea 59). La variable `nContador` sirve para diferenciar el primer envío de los demás y para determinar el número del segundo mensaje en adelante.

Nota: para simplificar el código, después del mensaje 9 todo se reinicia.

En la línea 53, se realiza la primera llamada para enviar datos a la cola, colocando el elemento al final:

```
053: xQueueSendToBackFromISR( xCola, (void
*)szTextoPrimerElemento, &xHigherPriorityTaskWoken);
```

El primer parámetro es el identificador de cola, `xCola`, es el identificador de cola. Indica a la función qué cola, entre las posibles colas, debe recibir los datos.

El segundo parámetro es el puntero genérico al elemento. En nuestro caso, esperamos un vector de byte (carácter) de 21 posiciones. Tenga en cuenta que siempre se pasa un puntero a la región de memoria que se va a pasar, pero la cola tiene su propia área de memoria para el elemento, con el tamaño dado (en este caso 21 bytes).

Lo que sucede es que se copian 21 bytes de la memoria del vector a la memoria del elemento. El puntero genérico `(void *)` no siempre es necesario, pero en nuestro caso destaca su naturaleza.

El tercer parámetro es apropiado para las funciones de API RTOS y es para el cambio de contexto (como en la función de semáforo en el listado 7) y en nuestro ejemplo se usa en la línea 65 para decidir si llamar a la función para forzar la evaluación del planificador .

Nota: La función `xQueueSendToBackFromISR` es una función de interrupción especial (termina en FromISR). Si vamos a enviar un elemento de una *task* , simplemente llame a la versión de la función `xQueueSendToBackFromISR` (que se ejemplificará en el siguiente ejemplo).

Una vez que los textos están en cola, alguien tiene que cuidarlos. Esto se realiza mediante la función de *task* creada, que se encuentra entre las líneas 72 y 86. Al igual que las *tasks* vistas hasta ahora, tiene un *loop* infinito llamado función de cola de lectura y la acción a tomar cuando se recibe cualquier elemento:

```
081:      // Espera indefinidamente hasta leer un elemento de la
cola
082:      xQueueReceive( xCola, &szTexto, portMAX_DELAY );
083:
084:     Serial.print(szTexto);
```

En la línea 82 tenemos la función call `xQueueReceive()` , que lee el elemento de la cola. El primer parámetro es el identificador de la cola y en el segundo debemos pasar un puntero a la variable donde debe colocarse una copia de los datos del elemento, en nuestro caso el texto que debe enviar el serial, con un tamaño compatible con el del elemento. En este caso, siempre debemos pasar la dirección, es decir, preceder la variable con el operador &.

En el tercer parámetro, que indica cuánto tiempo (en *ticks*) esperará un elemento, pasamos la constante `portMAX_DELAY` para que la función permanezca en esa línea indefinidamente, hasta que la interrupción ponga al menos un elemento en la cola. Este enfoque prepara la *task* , con la esperanza de manejar un nuevo elemento en la cola.

Ejercicio:

Se sugiere como ejercicio las siguientes mejoras y extensiones:

- Agreguemos una *task* entre la interrupción y la *task* que envía el texto;
- La interrupción ya no tratará el texto y se acortará;
- Midamos el tiempo aproximado entre una pulsación y otra;

(En el Apéndice 2 puede ver una posible solución a este ejercicio, pero le animo a que al menos intente reunir los conocimientos que ha adquirido hasta ahora y pueda resolver este ejercicio).

Para hacer esto debes:

- Cree una nueva cola con el tamaño del elemento de cola al tamaño de un valor que ahorre tiempo en *ticks* : `sizeof( TickType_t )`.
- Cambie la interrupción para leer el contador de ticks con la función `xTaskGetTickCountFromISR()` y envíelo a la cola y cree una *task* intermedia para manejar estos valores que crea una variable para conocer el valor del último contador recibido y otro para pasar un puntero a la función. ambos del tipo `TickType_t` . El tiempo se puede calcular de la siguiente manera:

```
nTiempoEnMs = (xAhora – xTiempoUltimaInterrupción) *
portTICK_PERIOD_MS;
```

- Esta *task* intermedia debería, usando `sprintf()` u otra función similar, escribir en la cola original (texto) algo así como “Botón: 2000 ms” (suponiendo que legara 2 segundos entre una pulsación y la otra. Tenga en cuenta que la segunda *task* no necesita sin ajuste, siempre que la longitud de 21 caracteres sea suficiente, en cuyo caso debe usar la función de carga normal:

```
xQueueSendToBack(xCola, szTexto, portMAX_DELAY);
```

Pruebe su programa observando que el tiempo se envía con mucha precisión ya que la resolución del contador de *ticks* es muy alta. La primera lectura mostrará *ticks* o el tiempo transcurrido desde el inicio de la ejecución y los demás mostrarán el tiempo entre lecturas.

Envío de marcos desde múltiples fuentes

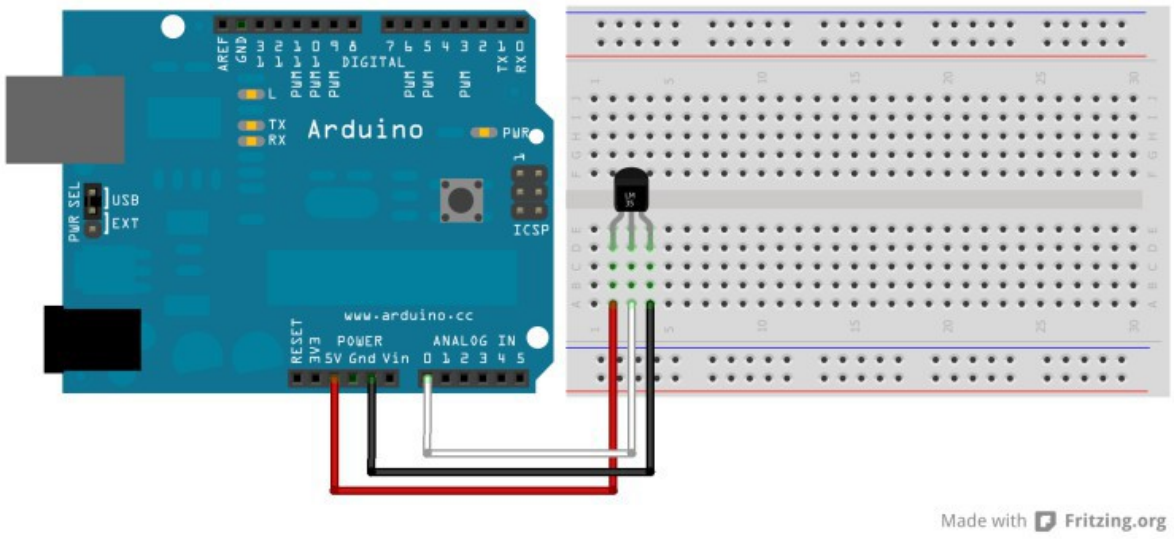
Para hacer las cosas más interesantes, trabajemos con una cola que tenga elementos que se agreguen desde más de una fuente, es decir, tendremos una *task* para leer los elementos de la cola y más de una *task* para poner valores en la cola.

Es cierto que el siguiente ejemplo no es una aplicación tan práctica, pero supongamos que tenemos una *display* LCD o TFT que muestra datos de más de una fuente:

- Un sensor de temperatura conectado a la entrada analógica;
- Un reloj interno simulado con tiempos de *task* (de segundo a segundo);
- El último texto recibido por la serie;

Para implementar esto, puede usar el sensor LM35 y seguir el tutorial escrito por Alan Motta y disponible en https://portal.vidadesilicio.com.br/lm35-medindo-temperatura-com-arduino (Lit 17/07/2018) .

Avanzando un poco las cosas, ensambla el siguiente circuito:



Fuente: https://portal.vidadesilicio.com.br/lm35-medindo-temperatura-com-arduino

Como sucedió antes, justo al comienzo del libro, no tiene que configurar el circuito (pero sería genial si lo hiciera) y dejar la puerta abierta o conectar un potenciómetro entre VCC y GND con el tercer terminal conectado al puerto analógico 0, simulando el sensor de temperatura. La tarea responsable de manejar este sensor leerá sus valores cada 500 ms y pondrá en cola la información de lectura, ya convertida a grados centígrados.

Para la segunda fuente crearemos una *task* que se detiene por un segundo y luego calcula el tiempo aproximado y lo envía a la cola.

Para la tercera fuente, nuestra *task* leerá el serial 0 (predeterminado) y tomará cada línea recibida limitando el texto a un cierto tamaño.

La última *task* en serie que se comunica con la pantalla(*display*). Realmente no implementaremos esto en este ejemplo, pero enviaremos dos líneas de 16 caracteres cada una, cada cambio de datos, simulando la actualización de una pantalla LCD de 2x16. Es un ejercicio para implementar hardware y comunicación con la pantalla LCD.

Antes de presentar el listado, puede ser interesante avanzar en el aspecto del elemento de la cola:

Hasta ahora trabajamos con el envío de texto y un valor numérico. Para enviar datos desde múltiples fuentes, utilizaremos una estructura con un elemento que identifica la fuente (y, por lo tanto, el tipo de información) y otro definido por una unión (*union*) (para interpretar una región de memoria común de diferentes maneras). Las fuentes se definieron por un tipo enumerado:

```
   // Tipo enumerado que indica qué información hay en el
elemento de la cola
   typedef enum {tiValorLecturaTemperatura, tiHoras,
tiTextoRecebido} TipoItem_t;
```

Y la estructura de otro tipo, que se usará para variables locales en tareas, parámetros de función y configuración de tamaño de elemento de cola:

```
   // Tipo de los tres tipos de datos (usando la unión) para
interpretar la memoria de una especie
   typedef struct {
    TipoItem_t Tipo;
    union {
     float Temperatura;
     struct {
```

```
      uint8_t Hora;
      uint8_t Minutos;
      uint8_t Segundos;
    } Horario;
    char Texto[13];
  } Datos;
} ItemFila_t;
```

El tipo `ItemFila_t` se verá en todo el código y ahora ya sabe cómo entenderlo de antemano.

Siga la lista, examínela y observe las diversas *tasks* , funciones auxiliares y uso de la cola. Para proporcionar referencias a la cola se resaltará para que pueda ver fácilmente los puntos donde se insertan los elementos al final de la cola y donde se eliminan los elementos.

Listagem 9:
LeituraTemperaturaRelogioTexto

```
001: #include <Arduino_FreeRTOS.h>
002: #include <queue.h>
003:
004: // Tipo enumerado que indica qué información hay en el elemento de la cola
005: typedef enum {tiValorLecturaTemperatura, tiHoras, tiTextoRecebido} TipoItem_t;
006:
007: // Tipo de los tres tipos de datos (usando la unión) para interpretar la memoria de una especie
008: typedef struct {
009:   TipoItem_t Tipo;
010:   union {
011:     float Temperatura;
012:     struct {
013:       uint8_t Hora;
014:       uint8_t Minutos;
015:       uint8_t Segundos;
016:     } Horario;
017:     char Texto[13];
018:   } Dados;
019: } ItemFila_t;
020:
```

```
021: //Identificador da fila unica
022: QueueHandle_t xCola;
023:
024: void TaskTomarLecturaTemp( void *pvParametros );
025: void TaskGenerarHoraActual( void *pvParametros );
026: void TaskRecepcaoSerial( void *pvParametros );
027: void TaskEnvio( void *pvParametros );
028:
029: int crearTasks(void)
030: {
031:     if(xTaskCreate(TaskLeituraTemp, (const portCHAR *)
"TaskLeituraTemp",  128, NULL, 2, NULL) == pdPASS)
032:       Serial.println("Task TaskLeituraTemp criada com
sucesso!");
033:     else
034:       return 0;
035:
036:     if(xTaskCreate(TaskHoraAtual, (const portCHAR *)
"TaskHoraAtual",  128, NULL, 2, NULL) == pdPASS)
037:
038:       Serial.println("Task TaskHoraAtual criada com
sucesso!");
039:     else
040:       return 0;
041:
042:     if(xTaskCreate(TaskRecepcaoSerial, (const portCHAR *)
"TaskRecepcaoSerial",  128, NULL, 2, NULL) == pdPASS)
043:       Serial.println("Task TaskRecepcaoSerial criada com
sucesso!");
044:     else
045:       return 0;
046:
047:     if(xTaskCreate(TaskEnvio, (const portCHAR *)
"TaskEnvio",
048:       128, NULL, 2, NULL) == pdPASS)
049:       Serial.println("Task TaskEnvio criada com
sucesso!");
050:     else
051:       return 0;
052:
053:     return 1;
054: }
055:
056: void setup() {
057:   BaseType_t xRet;
058:
059:   Serial.begin(9600);
```

```
060:
061:    // Cria a fila para as leituras de tmeperatura
062:    xCola = xQueueCreate(10, sizeof( ItemFila_t ));
063:   if( xCola )
064:   {
065:     Serial.println("Cola creada con éxito");
066:     if(!crearTasks())
067:     {
068:       Serial.println("Error al crear tasks");
069:     }
070:   }
071:   else
072:   {
073:     Serial.println("Error al crear la cola");
074:   }
075: }
076:
077: void loop() {
078:    //Monitorización de la flag dejada aquí
079: }
080:
081:
082: void TaskTomarLecturaTemp( void *pvParametros )
083: {
084:   ItemFila_t Item;
085:   Item.Tipo = tiValorLecturaTemperatura;
086:
087:   Serial.println("Inciando loop: TaskLeituraTemp");
088:
089:    //loop de task
090:   for( ;; )
091:   {
092:     int sensorValue = analogRead(A0);
193:
094:     Item.Datos.Temperatura =
(float(sensorValue)*5/(1023))/0.01;
095:
096:     xQueueSendToBack( xCola , &Item, portMAX_DELAY);
097:
098:     vTaskDelay(500 / portTICK_PERIOD_MS);
099:   }
100: }
101:
102: void tratarRelogio(ItemFila_t *pItem)
103: {
104:     pItem->Dados.Horario.Segundos++;
```

```
105:     if(pItem->Dados.Horario.Segundos == 60)
106:     {
107:       pItem->Dados.Horario.Segundos = 0;
108:       pItem->Dados.Horario.Minutos++;
109:
110:       if(pItem->Dados.Horario.Minutos == 60)
111:       {
112:         pItem->Dados.Horario.Minutos = 0;
113:         pItem->Dados.Horario.Hora++;
114:         if(pItem->Dados.Horario.Hora == 24)
115:         {
116:           pItem->Dados.Horario.Hora = 0;
117:         }
118:       }
119:     }
120: }
121:
122:
123: void TaskGenerarHoraActual( void *pvParametros )
124: {
125:   ItemFila_t Item;
126:   Item.Tipo = tiHoras;
127:   Item.Datos.Horario.Hora = 0;
128:   Item.Datos.Horario.Minutos = 0;
129:   Item.Datos.Horario.Segundos = 0;
130:
131:   Serial.println("Inciando loop: TaskRecepcaoSerial");
132:
133:    //loop de task
134:   for( ;; )
135:   {
136:     tratarRelogio(&Item);
137:
138:     xQueueSendToBack( xCola , &Item, portMAX_DELAY);
139:
140:      //espera un segundo
141:     vTaskDelay(1000 / portTICK_PERIOD_MS);
142:   }
143: }
144:
145: void TaskRecepcaoSerial( void *pvParametros )
146: {
147:   ItemFila_t Item;
148:   char carLido;
149:
150:   int tamano = 0;
151:
```

```
152:   Item.Tipo = tiTextoRecebido;
153:   Item.Datos.Texto[tamano] = '\0';
154:
155:   Serial.println("Inciando loop: TaskRecepcaoSerial");
156:
157:    //loop de task
158:   for( ;; )
159:   {
160:     if(Serial.available())
161:     {
162:       carLido = (char)Serial.read();
163:       if( carLido != '\n')
164:       {
165:         Item.Datos.Texto[tamano++] = carLido;
166:         Item.Datos.Texto[tamano] = '\0' ;
167:       }
168:
169:       if( (carLido == '\n') || (tamano ==
(sizeof(Item.Datos.Texto)-1)) )
170:       {
171:         xQueueSendToBack( xCola , &Item, portMAX_DELAY);
172:
173:         tamanho = 0;
174:         Item.Datos.Texto[tamano] = '\0';175:       }
176:     }
177:   }
178: }
179:
180:
181: void TaskEnvio( void *pvParametros ) {
182:   char EspejoDeDisplay[2][17] = {"00:00:00 |      ",
183:                                  "T: 000.0 |      "};
184:   ItemFila_t Item;
185:   int l,i;
186:   char *p;
187:
188:
189:   Serial.println("Comenzando el loop: TaskEnvio");
190:
191:    //loop de task
192:   for( ;; )
193:   {
194:      //Espera indefinidamente
195:     xQueueReceive( xCola , &Item, portMAX_DELAY );
196:
```

```
197:      //según el tipo de elemento, actualice el espejo de
display y envíe por comunicación en serie
198:     switch(Item.Tipo)
199:     {
200:       case tiValorLecturaTemperatura:
201:         sprintf(&EspejoDeDisplay[1][3], "%3.1f",
Item.Datos.Temperatura);
202:       break;
203:
204:       case tiHoras:
205:         sprintf(&EspejoDeDisplay[0][0], "%0.2d",
206:           Item.Datos.Horario.Hora);
207:         sprintf(&EspejoDeDisplay[0][3], "%0.2d",
208:           Item.Datos.Horario.Minutos);
209:         sprintf(&EspejoDeDisplay[0][6], "%0.2d",
210:           Item.Datos.Horario.Segundos);
211:       break;
212:
213:       case tiTextoRecebido  :
214:         //acomoda parte do texto em 6 caracteres em cima e
6 na parte de baixo.
215:
216:         //limpa
217:
218:         for(l=0; l<2; l++)
219:         {
220:           for(i=0, p = &EspejoDeDisplay[l][10]; i<6; i++,
p++) *p = ' ';
221:           *p = '\0';
222:         }
223:
224:         //copia
225:         for(l=0; l<2; l++)
226:         {
227:           p = &EspejoDeDisplay[l][10];
228:           for(i=0; i<6 && Item.Datos.Texto[i]; i++)
229:           {
230:             *p = Item.Datos.Texto[i];
231:             p++;
232:           }
233:
234:           *p = '\0';
235:         }
236:
237:       break;
238:     }
```

```
239:
240:
241:     Serial.println("----------------");
242:     Serial.println(EspejoDeDisplay[0]);
243:     Serial.println(EspejoDeDisplay[1]);
244:   }
245: }
```

Uf! Este listado se ha vuelto más grande. Pero vale la pena allí. No todo será comentado para hacer las cosas un poco más rápido...

En las líneas entre 29 y 54 tenemos una función para crear las *tasks* , devuelve 0 (falso) si alguno de ellos devuelve error y 1 (verdadero) si todo va bien. Esta función se llama función `setup()` .

La línea 62 crea la fila única y, si todo va bien, la función que crea las *tasks* se llama línea 66.

Ahora comentemos las funciones de *tasks* una por una.

Tarea de lectura del sensor de temperatura

El primero de ellos es la función `TaskTomarLecturaTemp` definida en la línea 82. En primer lugar, una variable local para el elemento, que solo será conocida por la *task* y que ya inicia el tipo de elemento como lectura de temperatura (`tiValorLecturaTemperatura`) en las líneas: 84 y 85 .

Dentro del *loop* de *tasks* , la entrada analógica A0 se lee en la línea 92 y el valor de temperatura se calcula y se asigna al elemento en la línea 94.

Esta lectura se agrega a la cola en la línea 96:

```
096:     xQueueSendToBack(xCola, &Item, portMAX_DELAY);
```

La dirección del elemento se pasa a la función que luego copia todos los bytes del elemento a la posición del nuevo elemento de la cola, que luego lo almacena individualmente. No hay relación con los datos en la cola y la variable instanciada en la línea 84. La cola mantiene todo aislado, incluso controlando la concurrencia, por lo que no ocurrirá que las *tasks* que leen elementos lean su valor antes de copiarlo y todo controles de cola configurados. No necesita proteger este código como una sección crítica o algo así, ya que la función API misma se encarga de ello. Incluso puede suceder tan pronto como se haya agregado este elemento, la *task* en espera de que se ejecuten los elementos y la *task* actual ya suspendida, especialmente si la *task* de lectura tenía una prioridad más alta que esta.

Luego se realiza un retraso de 500 ms, se suspende la *task* durante este tiempo y luego se repite toda la operación para una nueva lectura.

Tenga en cuenta que de esta manera tenemos un fragmento de código muy aislado, que hace lo que necesita y lo envía a otro punto, que ni siquiera conoce, una copia de los datos que se tratarán según sea necesario, sin tener que cambiarlos en este momento. (Nunca me canso de notar la belleza de este desacoplamiento).

Tarea de generación de tiempo actual

La segunda de las funciones es la función `TaskHoraAtual` definida a partir de la línea 123. D Al igual que la *task* anterior, instancia una variable local para el elemento, que será conocida solo por la *task* y ya inicia el tipo de elemento, esta vez indicando que es un elemento de tiempo (`tiHoras`). También inicia el elemento con todos los tiempos restablecidos. El tiempo actual se controlará en el elemento mismo, llamando a la función `tratarRelogio` , espaciada aproximadamente un segundo en la línea 136, dentro del *loop* principal.

¡Sólo un poco! ¿Cómo es que es? ¿Puedo llamar a una función desde la función de *task* ?

Sí. Ya lo hemos hecho llamando a las funciones de la API freeRTOS y a las funciones (o métodos) de arduino. Una cosa a tener en cuenta son las funciones que llamamos y las que escribimos (como ahora es nuestro caso) es que la función es reentrante, es decir, tiene código que se puede invocar desde más de una *task* a la vez, sin para que los datos se barajen [4] .

Es bastante cierto que esta función aún no es invocada por más de una *task* , pero tiene la condición completa de ser, ya que los únicos datos a los que accede es el parámetro pasado por la *task* en ejecución, que a su vez es una variable local en esta *task* , lo que hace que para que no haya posibilidad de problemas, porque si una segunda *task* comienza a ejecutarse mientras la función está en el medio, y esta tarea llama inmediatamente a la función, el valor pasado por parámetro será un puntero a otra variable, si es local La función de la *task* también garantizará la seguridad y la función será reentrante.

Además, la función de *task* sigue enviando un nuevo valor a la cola cada segundo a través de la llamada que agrega el nuevo valor de tiempo a la cola en la línea 138:

```
138:     xQueueSendToBack(xCola, &Item, portMAX_DELAY);
```

Notará que la llamada es idéntica a la de la línea 96 en la *task* discutida anteriormente. Esto se debe a que la variable del elemento tiene el mismo nombre (podría ser diferente) y la cola es la misma. Recuerde que tenemos tres *tasks* de producción como consumidor de datos. La gran diferencia son los datos enviados a la cola, que tiene un tipo diferente y tendrá que manejarse de manera diferente a la temperatura recibida o las lecturas de mensajes (que veremos a continuación).

Tarea para recibir de comunicación en serie

La tercera tarea se implementa en la función `TaskRecepcaoSerial` definida en la línea 145.

Entre las líneas 147 y 153, esta función crea dos variables para controlar la recepción realizada carácter por carácter (`carLido` y `tamano`) y, como las demás, tiene una variable para el elemento e intenta definir su tipo (`tiTextoRecebido`) e inicializar los datos con el contador. recepción despejada y con una cadena vacía (líneas 152 y 153).

En la línea 160, se busca información sobre el objeto en serie y, si es así, se lee y agrega al búfer del elemento en sí (línea 165) siempre que no sea un carácter de salto de línea (‘\ n').

En la línea 169 comprueba si el carácter \ n \ n 'ha llegado o si se ha utilizado toda la capacidad del *buffer* . En cualquier caso, agrega el elemento a la cola y reinicia el control de recepción.

Tenga en cuenta que después de copiar el contenido del elemento en la cola, se puede usar a continuación, como ocurre en la línea 166, donde el carácter nulo se coloca en la primera posición del *buffer* `Item.Datos.Texto` .

Nota: Esta recepción no es la mayor maravilla del mundo, porque si el usuario no ingresa tantos caracteres como sea necesario para llenar el *buffer* o no configura el terminal para enviar un '\ n', el elemento no se pondrá en cola, sino para nuestro propósito ya sirve [5] .

Tarea de lectura de la cola

Por fin!

Esta *task* , cuya función de implementación se llama `TaskEnvio()` y comienza en la línea 181, tiene como corazón leer los elementos de la cola en la línea 195 con la función `xQueueReceive()` y tratar el elemento de la línea 198.

Le propongo que examine la versión "comprimida" de este código a continuación:

```
194:      //Espera indefinidamente
195:     xQueueReceive( xCola , &Item, portMAX_DELAY );
196:
197:      //según el tipo de elemento, actualice el espejo de
visualización y envíe por comunicación en serie
198:     switch(Item.Tipo)
199:     {
200:       case tiValorLecturaTemperatura:
201:         sprintf(&EspejoDeDisplay[1][3], "%3.1f",
Item.Datos.Temperatura);
202:       break;
203:
204:       case tiHoras:
205:          // a la línea 210: Actualizar EspejoDeDisplay con
tiempo
211:       break;
212:
213:       case tiTextoRecebido  :
214:          // a la línea 235: EspejoDeDisplay  con texto
236:
237:       break;
238:     }
239:
240:
241:     Serial.println("----------------");
242:     Serial.println(EspejoDeDisplay[0]);
243:     Serial.println(EspejoDeDisplay[1]);
```

En la línea 195, la variable del artículo ingresada primero en la cola (o la más antigua aún no eliminada) se copia en la variable del artículo. Si no

hay elementos, la *task* se suspende porque el tiempo de espera para una lectura pasada en el tercer parámetro fue `portMAX_DELAY` .

Si pasamos otro valor (5 ticks, por ejemplo) tendremos que probar después del retorno de la función si devuelve `pdPASS` , en algo como:

```
 if(xQueueReceive( xCola , &Item, 5) == pdPASS)
 {
  //Según el tipo de elemento, actualiza el espejo ...
 switch(Item.Tipo)
  {
...
```

Pero este no es nuestro caso ...

Regresando: El elemento que leemos puede haber sido agregado por cada una de las tres *tasks* , por lo que para abordarlo usamos el marco de decisión de `switch case` , probando su tipo y escribiendo un bloque de código para cada tipo esperado.

Una vez que sepamos el tipo, podemos tomar el elemento de unión `Item.Datos` . Dada la información adecuada y usarla para actualizar la parte del espejo a la pantalla LCD 2x16 adecuada:

```
   Si tiValorLecturaTemperatura leerá desde
Item.Datos.Temperatura
   Si tiHoras leerá desde Item.Datos.Horario
   Si tiTextoRecebido, leerá de Item.Datos.Texto
```

Finalmente, enviará este espejo a la serie simulando la actualización de la pantalla (*display*)(líneas 241 a 143) y esperará el siguiente elemento nuevamente. Como esta *task* no tiene temporizaciones, incluso teniendo la misma prioridad que otras *tasks* , manejará rápidamente la cola, siempre estando cerca de cero.

Como este código es interesante y quiero ofrecerle un ejercicio práctico, aparece nuevamente en el apéndice, pero de manera más compacta y sin líneas.

Propostas de exercícios [6] :

- Ejecute el programa de archivos adjuntos tal como está;

- Cree proyectos con fuentes originales y realice cambios y experiencias gratuitas. Si utiliza el ejemplo con la tecla aquí y agrega un nuevo tipo de entrada, enviado por la interrupción, ¿cambia un carácter de visualización con un carácter diferente cada vez que se activa la tecla?
- Cree otro proyecto con las fuentes e intente disminuir el tiempo dentro del bucle TaskReadTemp de 500 / portTICK_PERIOD_MS a 400 / portTICK_PERIOD_MS, 300 / portTICK_PERIOD_MS, 100, 50 / portTICK_PERIOD_MS, y finalmente 1 (una marca) y compruebe cómo se envía datos del reloj en relación con el envío de datos de temperatura (las cosas pueden ponerse un poco raras).
- Cambie las prioridades de las tareas de producción (en la función `crearTasks()`) a un valor inferior (de 2 a 1) y mantenga la de la tarea de consumo (`TaskEnvio`) con el valor 2. De esta forma, este último tendrá mayor prioridad que los demás y tomará el control cada vez que se agregue un elemento a la cola. Comprueba cómo va la ejecución.

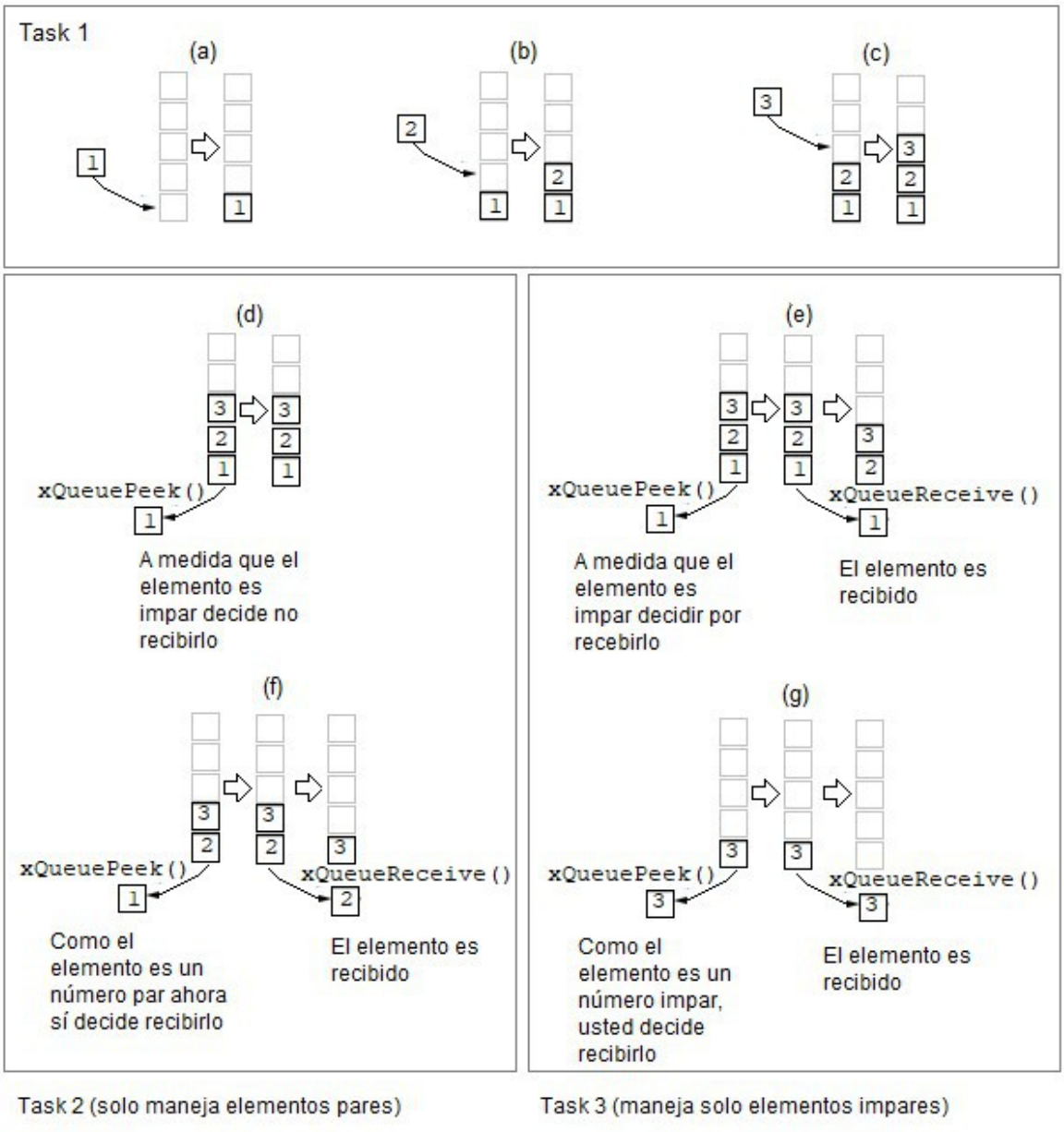
Otras opciones de acceso a la cola

Hasta ahora hemos utilizado la función `xQueueSendToBack()` , su variación para las interrupciones (`xQueueSendToBackFromIRS()`), tanto para insertar elementos al final de la cola como la función `xQueueReceive()` para eliminar elementos de la cola. Sin embargo, tenemos otras opciones de función en la API de freeRTOS, entre las cuales utilizamos cuatro de ellas:

- `xQueuePeek()`
- `xQueueOverride()`
- `xQueueSendToFront()`
- `xQueueReset()`

Dada una ojeada en el tema sin sacarlo

Hay situaciones en las que echar un vistazo al artículo y decidir si sacarlo de la cola o no es una buena idea. Puede decidir dejarlo en la cola (para que otra *task* lo elimine) si no está dirigido a usted o si tiene un tipo de datos que no puede manejar. Esto se ilustra en la siguiente figura:



Nota: Realice los pasos de (a) a (g) y cada paso indica la fila anterior y posterior del paso, separándola con una flecha de izquierda a derecha.

Passos:

(a) LaTask1 agrega a la cola vacía el elemento con valor 1;

(b) La Task1 se agrega a la cola, que ya tiene el elemento de valor 1, el elemento de valor 2;

(c) La Task1 agrega a la cola que ya tiene los elementos de valor 1 y 2, el elemento de valor 3;

(d) La Task2 mira el elemento con `xQueuePeek()` y "ve" que no es para ella porque es extraño y ya no lo elimina.

(e) La Task3 espía el elemento con `xQueuePeek()` y "ve" que depende de ella eliminar el elemento de la cola, ya que es extraño. Eliminándolo con una llamada a `xQueueReceive()` .

(f) La Task2 espía que el elemento "ve" que depende de ella eliminar el elemento de la cola, porque es uniforme. Eliminándolo con luego de la cola.

(g) La Task3 "ve" que el elemento es extraño, lo elimina de la cola y lo deja vacío.

Codifiquemos un poco más ...

Para ilustrar esto, hemos modificado el código en el Listado 9, creando una *task* más que consumirá la cola, pero solo para el tipo de datos de tiempo, que ya no se envía en el espejo de la pantalla (*display*) y solo se prueba para simular un sistema de alarma.

Ya no repetiremos la lista completa, sino la función `TaskEnvio()` modificada y la nueva función `TaskTrataHorario()` , que corresponde al nuevo código de *task* . Ya debería poder cambiar el código anterior para declarar la función y crear la nueva *task* .

Examine el código de la función anterior (con los cambios en negrita) y la nueva función y lea cuidadosamente la siguiente explicación:

```
001: void TaskEnvio( void *pvParametros ) {
002:    char EspejoDeDisplay[2][17] = {"00:00:00 |      ",
003:                                   "T: 000.0 |      "};
004:
005:    ItemFila_t Item;
006:    int l,i;
007:    char *p;
008:    int usado = 0;
009:
010:    Serial.println("Comenzando el loop: TaskEnvio");
011:
012:    //loop de task
013:    for( ;; )
014:    {
015:       //Espera indefinidamente
016:       xQueuePeek( xCola, &Item, portMAX_DELAY );
017:       usado = 1;
```

```
018:      //de acuerdo con el tipo de elemento, actualiza el espejo de la pantalla y lo envía a través de comunicación en serie o ignore el elemento
019:      switch(Item.Tipo)
020:      {
021:        case tiValorLecturaTemperatura:
022:          sprintf(&EspejoDeDisplay[1][3], "%3.1f", Item.Datos.Temperatura);
023:         break;
024:
025:        case tiHoras:
026:          // No se hace nada excepto marcar elemento rechazado
027:          usado = 0;
028:        break;
029:
030:        case tiTextoRecebido:
031:          //acomoda parte del texto en 6 caracteres en la parte superior y 6 caracteres en la parte inferior.
032:          //limpio
033:         for(l=0; l<2; l++)
034:         {
035:           for(i=0, p = &EspejoDeDisplay[l][10]; i<6; i++, p++) *p = ' ';
036:           *p = '\0';
037:         }
038:
039:          //copia
040:         for(l=0; l<2; l++)
041:         {
042:           p = &EspejoDeDisplay[l][10];
043:           for(i=0; i<6 && Item.Datos.Texto[i]; i++)
044:           {
045:             *p = Item.Datos.Texto[i];
046:             p++;
047:           }
048:           *p = '\0';
049:         }
050:        break;
051:
052:      }
053:
```

```
      if( usado)
      {
          // para eliminar y enviar texto de manera efectiva a
través de la comunicación en serie en este punto
          xQueueReceive( xCola, &Item, portMAX_DELAY );

          Serial.println("----------------");
          Serial.println(EspejoDeDisplay[0]);
          Serial.println(EspejoDeDisplay[1]);
      }
      else
      {
        // Retrasos por tic para inducir el movimiento a otra
task
          vTaskDelay(1);
      }
    }
}

void TaskTrataHorario( void *pvParametros ) {
   ItemFila_t Item;

   Serial.println("Incio de la ejecución del loop:
TaskTrataHorario");
    //loop de task
   for( ;; )
   {
      //Espera indefinidamente
     xQueuePeek( xCola, &Item, portMAX_DELAY );
      if( Item.Tipo != tiHoras)
     {
        // Retrasos por una unidad mínima de tiempo para inducir
el pase ...
         vTaskDelay(1);
       continue;
     }

     xQueueReceive( xCola, &Item, portMAX_DELAY );
      //  Enviar un nuevo elemento como texto
      // alarmas de señalización
     if( ( Item.Datos.Horario.Hora == 0) &&
```

```
091:          ( Item.Datos.Horario.Minutos == 0) &&
092:          ( Item.Datos.Horario.Segundos == 10) )
093:      {
094:        Item.Tipo = tiTextoRecebido;
095:        sprintf(Item.Datos.Texto, "Alarme10s !");
096:      }
097:      else if( ( Item.Datos.Horario.Hora == 0) &&
098:               ( Item.Datos.Horario.Minutos == 2) &&
099:               ( Item.Datos.Horario.Segundos == 0) )
100:      {
101:        Item.Tipo = tiTextoRecebido;
102:        sprintf(Item.Datos.Texto, "Alarme120s !");
103:      }
104:      else
105:        continue;
106:
107:      xQueueSendToBack(xCola, &Item, portMAX_DELAY);
108:    }
109: }
```

Antes de pasar la explicación, es noticia que esta fuente completa (pero la original en portugués), con las modificaciones propuestas en el listado 9 está disponible en github, en el enlace: https://github.com/maxback/multitarefa_na_pratica/tree/master/LeituraTemperaturaRelogioTextoDuasTasks y también el código que implementa el ejemplo de separación de números pares e impares en https://github.com/maxback/multitarefa_na_pratica/tree/master/SeparaParesDeImpares

Explicación

Dado que las indicaciones de numeración comenzaron con 1, comenzamos en la línea 1 con la función `TaskEnvio()` preexistente y agregamos la variable utilizada en la línea 8, que tendrá como objetivo indicar después de cada elemento leído de la cola si se trató o no. Si se maneja, se eliminará

efectivamente, de lo contrario, simplemente se ignorará y la *task* no hará nada con el elemento.

Dentro de for, en la línea 16, a la luz de la luna de la función de recepción a la que estamos acostumbrados, puede ver una llamada a la función `xQueuePeek()` . Esta función tiene los mismos parámetros que la anterior, pero con un comportamiento diferente: lee el elemento pero no se retira, permitiendo que su software decida si eliminarlo o dejar las cosas como están.

En la línea 17, la variable utilizada obtiene el valor verdadero en la línea 19, tenemos el c `switch case` de antes. Si los tipos son `tiValorLecturaTemperatura` y `tiHoras` , el elemento se tratará como antes. Nada cambió. Sin embargo, en la línea 25, el tipo `case tiTextoRecebido` para el tipo de texto recibido solo marca como falso y no trata el elemento.

En la línea 54, la variable `usado` se usa para decidir recibir el artículo (y finalmente eliminarlo de la cola en la línea 57) y enviar el espejo de la pantalla(*display*) por serie si se usa tiene un valor verdadero.

Si es falso, se llama a la función `vTaskDelay()` para cronometrar un tic y darle a la otra *task* la oportunidad de ejecutarse (simplemente induciéndolo, ya que el programador puede interrumpirlo debido al *time-slice*).

Esta *task* está hecha, a saber: maneja dos tipos de elementos y deja el otro para otra *task* , que es la *task* implementada por la función `TaskTrataHorario()` , que comienza en la línea 71.

La función `TaskTrataHorario()` es similar a la anterior, pero sin un `switch case` , ya que el interés está en un solo tipo, en lugar de la prueba `if()` de la línea 80, que multiplica y regresa al siguiente para la `for()` , usando `continue` en la línea 84.

En la línea 241, el artículo se retira de la cola para hacer espacio allí mismo y luego se evaluará en las pruebas de las líneas 244 y 251 que prueban el tiempo de 10 segundos y 120 segundos (2 minutos) respectivamente. Cada prueba de estos, si es verdadera, edita la propia variable del elemento, "reciclándola" para otro propósito: definir el elemento `tiTextoRecebido` y ensamblar y copiar un texto, simulando una recepción en serie para mostrar un texto indicativo de la alarma.

Si ninguno de estos `ifs` ejecuta, se llama al comando `continue` , para pasar la siguiente iteración del *loop* .

En la línea 261, el elemento de texto se envía al final de la cola y será procesado por la *task* `TaskEnvio` poco después de procesar cualquier elemento agregado por las otras *tasks* .

Sobrescribir un elemento

En algunos casos, podemos usar una cola con la capacidad de un solo elemento para pasar información a una *task* que tendrá que ser manejada. Pero, ¿qué hacer si la *task* aún no ha procesado la información y el evento ya se ha repetido? Otra pregunta: ¿Qué pasa si una vez que no ha sido tratado, ya no es bueno tratarlo y es más importante tratar la nueva información ahora?

En estos casos, el tratamiento más apropiado puede ser sobrescribir el contenido del elemento.

¿Alguna vez ha tenido a alguien abriendo una ventana y le tomó mucho tiempo abrirla, y cuando todavía está yendo a la ventana se dio cuenta de que sería mejor abrir la puerta? Sobrescribe un comando con el otro. Ahora, otra situación es cuando tenemos una cola de varios elementos y desea agregar un nuevo elemento al final, pero si no fuera posible, preferiría sobrescribir este elemento que suspender su *task* esperando liberar espacio en la cola.

Estos son dos casos en los que la función `xQueueOverride()` se puede utilizar para sobrescribir un elemento en una cola. Se usará pronto en el libro y puede ver su uso en la práctica.

Saltarse la cola

Hasta ahora hemos trabajado con colas agregando elementos siempre al final de la cola, es decir, el viejo sistema FIFO (*First In First Out :* Primero en entrar, primero en salir). Pero para una multitud de situaciones que pueden surgir, es posible que desee insertar el elemento al comienzo de la cola para tratarlo lo antes posible.

Citaré un ejemplo: en la lista anterior, que en realidad es solo una parte de la lista, modificada para tener una *task* más, tuve la tentación de poner el nuevo elemento de texto en caso de una alarma al comienzo de la cola, lo que sería equivalente a reemplazar el se ha eliminado con el tiempo detectado como alarma por otro con el texto, alertando al usuario de la alarma lo antes posible.

Simplemente cambie la llamada de la línea 107 de:

```
xQueueSendToBack(xCola, &Item, portMAX_DELAY);
```

a:

```
xQueueSendToFront(xCola, &Item, portMAX_DELAY);
```

Este código hará que el texto de la alarma se procese lo antes posible, ya que los elementos en espera se desplazarán hacia atrás, y este elemento ocupará la primera posición, que será la próxima en la cola.

Reiniciar la cola(reset)

Otra característica interesante es poder restablecer la cola, es decir, borrar la cola si ocurre algún evento, cuya evaluación da como resultado la decisión de que los datos en la cola ya no se procesen, como presionar un botón de cancelación.

En este caso, simplemente se puede llamar a la función `xQueueReset()`, pasando el identificador de la misma. Otro ejemplo es una *task* que tiene que leer una cola de paquetes para ser enviados a un periférico por comunicación en serie y otra de paquetes capturados a través de DMA o recibir interrupción. Puede ocurrir en cierto momento que lo que se recibió (y aún no se ha tratado) ya no importa, por lo que puede borrar la cola de datos recibidos para una nueva secuencia de envío.

En este caso, la cola de datos para enviar puede tener paquetes y comandos mixtos, y uno de estos comandos es borrar la cola de recepción (`"cmd:reset_rx"`), como se muestra en el siguiente fragmento de código:

```
01:   for (;;)
02:   {
03:     if(xQueueReceive(xColaEnvio, szBufferTx,
04:       portMAX_DELAY) == pdPASS)
05:     {
06:       bPurgeRx = strcmp(szBufferTx, "cmd:reset_rx") == 0;
07:
08:       if(bPurgeRx)
09:       {
10:         xQueueReset(xColaRecepcao);
11:         continue;
12:       }
13:       // Manejo si el paquete se enviará u otro comando
14:     }
15:   }
```

Más C++

Presentamos una idea de usar C ++ usando clases para encapsular llamadas FreeRTOS, con constructores que asignan la cola y los métodos de operador y sobrecargas. De esta manera, aquellos que conocen C ++ y prefieren usarlo pueden tener algo de inspiración para sus propias clases.

Listagem 10: Definición de clase CanalEntreTasks

```
001: #ifndef _CANALENTRETASKS_H_
002: #define _CANALENTRETASKS_H_
003:
004: #include "funcoes_apoio.h"
005: /**
006: * Clase que implementa un canal tipo golange y encapsula una
cola
007: * CanalEntreTasks<T>
008: */
009: template <typename T>
010:
011: class CanalEntreTasks {
012: public:
013:
014:    CanalEntreTasks(uint16_t pnTamanho):
015:      nTempoTimeoutMs(0), bResultadoOperacao(false)
016:    {
017:          xCola = xQueueCreate(pnTamanho, sizeof(T));
018:    };
019:
020:    CanalEntreTasks(uint16_t pnTamanho, const char *pszNome):

021:      CanalEntreTasks(pnTamanho)
022:    {
023:          setNome(pszNome);
024:    };
025:
026:    ~CanalEntreTasks() {
```

```
027:    if(xCola != NULL)
028:        vQueueDelete( xCola );
029:    };
030:   //devuelve el resultado de la última operación
031:    bool getResultadoOperacao(void) const
032:    {
033:      return bResultadoOperacao;
034:    }
035:    //Exponer el identificador de cola FreeRTOS
036:    QueueHandle_t getFilaInterna()
037:    {
038:      return xCola;
039:    }
040:    void setNome(const char *pszNome) //Establecer un nombre
041:    {
042:      strcpy(szNome, pszNome);
043:    };
044:    //Devuelve el nombre (generalmente para uso de
depuración)
045:    char * getNome(void)
046:    {
047:      return szNome;
048:    };
049:    //Define un tiempo de espera para la comunicación.
050:    void setTempoTimeoutMs(const int pnTempoTimeoutMs)
051:    {
052:      nTempoTimeoutMs = pnTempoTimeoutMs;
053:    };
054:   //Envía algo a la cola
055:    void enviar(const T &pxValor, bool pbTimeOut = false)
056:    {
057:      bResultadoOperacao = xQueueSendToBack( xCola, &pxValor,

058:        pbTimeOut ? pdMS_TO_TICKS(nTempoTimeoutMs) :
portMAX_DELAY ) == pdPASS;
059:    };
060:
```

```
061:    T receber(bool pbTimeOut = false) {
062:      T xValor;
063:       bResultadoOperacao = xQueueReceive( xCola, &xValor,
064:         pbTimeOut ? pdMS_TO_TICKS(nTempoTimeoutMs) :
portMAX_DELAY ) == pdPASS;
065:
066:       return xValor;
067:    };
068:    //Obtén, si te sales de la cola, el siguiente elemento
069:    T ver() {
070:      T xValor;
071:       bResultadoOperacao = xQueuePeek( xCola, &xValor,
portMAX_DELAY ) == pdPASS;
072:
073:       return xValor;
074:    };
075:
076: //Sobrescribir el elemento
077: void sobrescrever(T &pxValor) {
078:      T xValor;
079:       bResultadoOperacao = xQueueOverwrite( xCola, &pxValor
) == pdPASS;
080:
081:    };
082:
083: private:
084:    QueueHandle_t xCola;
085:    char szNome[33];
086:    int nTempoTimeoutMs;
087:    bool bResultadoOperacao;
088: };
089:
090:
091: // para permitir encadenar el operador << (escribir sin
tiempo de espera, bloquear siempre que no haya espacio)
092: // Esto es equivalente a llamar a CanalEntreTasks<T>::enviar
(T xValue);
093: template <typename T>
094: CanalEntreTasks<T>& operator<<(CanalEntreTasks<T> &c, T
pxValor)
```

```
095: {
096:     c.enviar(pxValor);
097:     return c;
098: }
099:
100: // para permitir encadenar el operador >> (leer sin tiempo de espera, bloques hasta que no se proporcionen datos) para recibir
101: // Equivalente a llamar a T CanalEntreTasks<T>::receber();
102: template <typename T>
103: CanalEntreTasks<T>& operator>>(CanalEntreTasks<T> &c, T &pxValor)
104: {
105:     pxValor = c.receber();
106:     return c;
107: }
108:
109: //para permitir encadear o operador > para ler (peek) o primeiro item que sairia
110: //Equivale a chamar T CanalEntreTasks<T>::ver();
111: template <typename T>
112: CanalEntreTasks<T>& operator>(CanalEntreTasks<T> &c, T &pxValor)
113: {
114:     pxValor = c.ver();
115:     return c;
116: }
117:
118: //para permitir encadear o operador < (override) para sobrescrever o ultimo elemento
119: //Equivale a chamar CanalEntreTasks<T>::sobrescrever(T &xValor);
120: template <typename T>
121: CanalEntreTasks<T>& operator<(CanalEntreTasks<T> &c, T &pxValor)
122: {
123:     c.sobrescrever(pxValor);
124:     return c;
```

```
125: }
126: #endif
127:
```

En la línea 14 tenemos el constructor de clase que recibe como parámetro el tamaño de la clase y en la línea 20 tenemos otra opción de constructor que además del tamaño, también recibe un texto para el nombre del canal, útil para *debug* el objeto que lo diferencia de otros llamando al método `getNome()` definido en la línea 45.

Consulte el Apéndice A.4 - Ejemplos de métodos de llamada CanalEntreTasks donde se realizan llamadas de muestra y también el enlace a github con el uso real de esta clase.

Capítulo 6 - Notificación entre tareas

Comunicación entre tareas con notificaciones.

Hasta ahora usamos semáforos o colas para comunicar eventos y datos entre *tasks* o entre interrupciones y *taks* . En estos enfoques, necesitamos crear objetos y así asignar más memoria de ambos tipos de objetos con una o más posiciones (en el caso de la cola).

Sin embargo, podemos aprovechar la estructura de datos ya creada para cada *tasks* para enviarle mensajes notificándoles a nivel de bit o de valor.

En la práctica, utilizamos una función fuera de la *task* , que necesita conocer su identificador, para cambiar el valor asociado con la notificación sobrescribiéndola, incrementándola o configurando bits específicos.

Todo esto se hace atómicamente, protegiendo secciones críticas y lo más importante, relativamente simple para el programador.

Dentro de la *task* está comprobar si ha llegado una notificación, como leer una cola, pero un poco más complejo.

Notificación de nivel de bit

En nuestro ejemplo trataremos las interrupciones externas 0 y 1 y enviaremos el valor de una estructura en la parte menos significativa del valor de notificación a la interrupción externa 0 y la parte más significativa para la interrupción 0. Dado que el valor es de 32 bits, cada estructura Puede ocupar hasta 16 bits.

Esta estructura contiene la ID, el tiempo en *ticks* de interrupción y un bit de *flag* para facilitar la detección del evento al leer el valor, como se muestra a continuación:

```
//a propósito para mapear 16 bits
typedef union {
struct {
uint16_t nFlagEventoOcurrido     : 1;
uint16_t nID                     : 4;
uint16_t nTiempoEntreOcurrencias : 11;
};
uint16_t nValor;
} datos_lectura_boton_t;
```

La estrategia de mapeo de bits se utilizó para garantizar que todo encaje en 16 bits. El elemento correspondiente al tiempo (`nTiempoEntreOcurrencias`) se vio afectado ya que se ajustará a un valor de solo 11 bits, lo que le permitiría contar intervalos de máximo 30 segundos entre interrupciones, pero en general toda esta estructura sirve como un ejemplo de la posibilidad de pasar datos. complejo en el evento.

El siguiente es el código de ejemplo con dos interrupciones que envían eventos a una *task* que cambia el tiempo de parpadeo del LED correspondiente a la tecla de acuerdo con el tiempo entre disparadores y también envía el valor de lectura a través del puerto de comunicación en serie, para que podamos seguir mejor lo que sucede.

Listagem 11: NotificacaoTasksDoisLEDs

```
001: #include <Arduino_FreeRTOS.h>
002: #include <task.h>
003:
004: //Estructura para almacenar datos sobre ocurrencias de
interrupción (16 bits)
005: typedef union {
006: struct {
007:           uint16_t nFlagEventoOcurrido     : 1;
008:           uint16_t nID                     : 4;
009:           uint16_t nTiempoEntreOcurrencias : 11;
010:         };
011:         uint16_t nValor;
012: } datos_lectura_boton_t;
013:
014: //Mapa de bits con el conjunto de bits nFlagEventoOcurrido
para usar en máscaras
015: #define BOTON_LECTURA_16_BIT_FLAGEVENTO1_OCURRIO
((uint16_t)0x0001)
016:
017: #define BOTON_LECTURA_16_BIT_FLAGEVENTO2_OCURRIO \
(((uint32_t)BOTON_LECTURA_16_BIT_FLAGEVENTO1_OCURRIO)<<16)
018:
019: //Definición de pin LED
020: #define LED_PIZZARRA LED_BUILTIN
021: #define LED_ADICIONAL 12
022:
023: static TaskHandle_t gxTaskNotificacion = NULL;
024:
025: static void EXTI0_IRQHandler(void);
026: static void EXTI1_IRQHandler(void);
027:
028: void setup()
029: {
030:   Serial.begin(9600);
031:
032:   pinMode(LED_PIZZARRA, OUTPUT);
033:   pinMode(LED_ADICIONAL, OUTPUT);
034:
```

```
035:   xTaskCreate(taskBotonUsuarioFunc, "taskBotonUsuarioFunc",
036:     128, NULL, 2, &gxTaskNotificacion );
037:
038:   attachInterrupt(0, EXTI0_IRQHandler, RISING);
039:   attachInterrupt(1, EXTI1_IRQHandler, RISING);
040: }
041:
042: void loop()
043: {
044:     //nada que hacer
045: }
046:
047: void EXTI0_IRQHandler(void)
048: {
049:   static TickType_t xUltimoTiempoInterrupcion = 0;
050:   TickType_t xAhora;
051:   BaseType_t xHigherPriorityTaskWoken = pdFALSE;
052:   uint32_t ulValorTaskNotificion;
053:   datos_lectura_boton_t lectura;
054:
055:   xAhora = xTaskGetTickCountFromISR();
056:
057:   lectura.nID = 0;
058:
059:   lectura.nTiempoEntreOcurrencias = xAhora -
xUltimoTiempoInterrupcion;
060:   lectura.nFlagEventoOcurrido = 1;
061:
062:   xUltimoTiempoInterrupcion = xAhora;
063:
064:   if(gxTaskNotificacion)
065:   {
066:     ulValorTaskNotificion = lectura.nValor;
067:     ulValorTaskNotificion <<= 0;
068:     ulValorTaskNotificion &= 0x0000FFFF;
069:
070:     //Envia uma notifica??o diretamente para a task indicada
por gxTaskNotificacion
071:     xTaskNotifyFromISR( gxTaskNotificacion,
072:       ulValorTaskNotificion, /* ulValue */
```

```
073:       eSetBits, /* eAction parameter. */
074:       &xHigherPriorityTaskWoken );
075:     }
076:
077:     if( xHigherPriorityTaskWoken )
078:     {
079:       taskYIELD ();
080:     }
081: }
082:
083: void EXTI1_IRQHandler(void)
084: {
085:   static TickType_t xUltimoTiempoInterrupcion = 0;
086:   TickType_t xAhora;
087:   BaseType_t xHigherPriorityTaskWoken = pdFALSE;
088:   uint32_t ulValorTaskNotificion;
089:
090:   datos_lectura_boton_t lectura;
091:   xAhora = xTaskGetTickCountFromISR();
092:   lectura.nID = 1;
093:
094:   lectura.nTiempoEntreOcurrencias = xAhora -
xUltimoTiempoInterrupcion;
095:
096:   lectura.nFlagEventoOcurrido = 1;
097:
098:   xUltimoTiempoInterrupcion = xAhora;
099:
100:   if(gxTaskNotificacion)
101:   {
102:     ulValorTaskNotificion = lectura.nValor;
103:     ulValorTaskNotificion <<= 16;
104:     ulValorTaskNotificion &= 0xFFFF0000;
105:
106:     //Enviar una notificación directamente a la tarea
indicada por gxTaskNotificacion
108:     xTaskNotifyFromISR( gxTaskNotificacion,
109:       ulValorTaskNotificion, /* ulValue */
110:       eSetBits, /* eAction parameter. */
111:       &xHigherPriorityTaskWoken );
```

```
  }

  if( xHigherPriorityTaskWoken )
  {
    taskYIELD ();
  }
}

void taskBotonUsuarioFunc( void *pvParameters )
{
  union {
    uint32_t ulLecturas;
    datos_lectura_boton_t lectura[2];
  } xDatos;

  char szMsgLectura[100];
  BaseType_t xTempos[2] = {portMAX_DELAY, portMAX_DELAY};
  int LEDs[2] = {LED_PIZZARRA, LED_ADICIONAL};
  int contador = 0;

  for (;;)
  {
    //parpadea LED a menos que no tenga tiempo
    for(int i=0; i<2; i++)
    {
      if(xTempos[i] == portMAX_DELAY) continue;

      digitalWrite(LEDs[i], contador&0x01 ? HIGH : LOW);
      vTaskDelay( xTempos[i] );
    }

    contador++;

    //notificación de espera con eventos
    xTaskNotifyWait(0, //ulBitsToClearOnEntry
      0xFFFFFFFF, //ulBitsToClearOnExit
      &xDatos.ulLecturas, //pulNotificationValue
      portMAX_DELAY);

    for(int i=0; i<2; i++)
```

```
152:     {
153:       if(!xDatos.lectura[i].nFlagEventoOcurrido)
154:         continue;
155:
156:       //ajustar el tiempo de LED correspondiente
157:       xTempos[i] =
xDatos.lectura[i].nTiempoEntreOcurrencias;
158:
159:       sprintf(szMsgLectura, "Evento ID %d, Tiempo: %d ms",
xDados.lectura[i].nID,
xDatos.lectura[i].nTiempoEntreOcurrencias);
160:
161:
162:       Serial.println(szMsgLectura);
163:     }
164:   }
165: }
```

Ahora ya estás más familiarizado con este tipo de código. Por lo tanto, nos centraremos en enviar la notificación y su devolución. El envío es muy similar entre las dos interrupciones, por lo que mostraremos el más complejo, el de la interrupción externa 1:

```
100:   if(gxTaskNotificacion)
101:   {
102:     ulValorTaskNotificion = lectura.nValor;
103:     ulValorTaskNotificion <<= 16;
104:     ulValorTaskNotificion &= 0xFFFF0000;
105:
106:     //Enviar una notificación directamente a la tarea
indicada por la gxTaskNotificacion
108:     xTaskNotifyFromISR( gxTaskNotificacion,
109:       ulValorTaskNotificion, /* ulValue */
110:       eSetBits, /* eAction parameter. */
111:       &xHigherPriorityTaskWoken );
112:   }
```

En la línea 100 se prueba si el identificador de *task* de destino se estableció realmente.

La variable de lectura, declarada en la línea 90 (rellenada en líneas posteriores) se copia a la variable `ulValorTaskNotificion` (aunque esta copia podría evitarse). Luego se desplaza 16 bits hacia la izquierda (ya que estos bits se establecerán en la notificación).

En la línea 104 se hace una máscara solo para reforzar esto, ya que el desplazamiento a la izquierda despeja los bits de todos modos. Este es mi estilo, aunque en un programa de alto rendimiento ejecutando instrucciones innecesarias.

La línea 108 finalmente llama a la función de notificación `xTaskNotifyFromISR()` , que pasa en su primer parámetro el identificador de la *task* de destino.

En la línea 109, el valor se pasa a la notificación y en la línea 110, la operación que se debe hacer con este valor (`eSetBits`), que indica que los bits 1 del valor pasado deben establecerse y los demás deben mantenerse como están.

Es esta operación la que permite que cada interrupción envíe valores en su porción de 16 bits del valor. Este mecanismo podría usarse para hasta 32 elementos de interrupción diferentes si cada uno se limita a establecer uno de los bits. Por lo tanto, es posible darse cuenta con el mecanismo oeste es versátil y potente.

Ahora repasemos cómo se ve el recibo:

```
145:    //esperar notificación con eventos
146:    xTaskNotifyWait(0, //ulBitsToClearOnEntry
147:      0xFFFFFFFF, //ulBitsToClearOnExit
148:      &xDatos.ulLecturas , //pulNotificationValue
149:      portMAX_DELAY);
```

En la *task* , en la línea 122, se define una `union` para poder tratar ambas partes del valor como una matriz `xDatos.lectura[]` , lo que permite atravesar muy convenientemente ambas partes probando el bit de bandera que indica que la notificación tiene datos escritos. en esa parte del valor. Estas son las *flags* `xDados. lectura[0].nFlagEventoOcurrido e xDados.lectura[1].nFlagEventoOcurrido` ;

La llamada para leer el valor notificado se llama en la línea 146. Tenga en cuenta que, a diferencia de otras funciones, en este caso no es posible

especificar de dónde se leen los datos, ya que siempre serán de la *task* en ejecución.

En el primer parámetro podríamos pasar un valor con los bits establecidos a 1, que indica qué bits ya se restablecerían al ingresar a la función. Se pasa un valor de 0, lo que indica que no se debe restablecer ningún bit.

En el segundo parámetro, indicamos los bits que deben establecerse en el valor de notificación de la *task* tan pronto como regrese la función. En nuestro caso, no queremos dejar nada allí (sino solo en la copia del valor original para `xDatos.ulLecturas`). Por eso se pasa el valor`0xFFFFFFFF` . De esta forma, todos los bits se borran y las notificaciones nuevas se pueden grabar con éxito.

En la línea 148 se pasa el valor a la variable de lectura ya mencionada, que será inspeccionada por la fuerza de la línea 151, que atraviesa ambas partes y pruebas (en la línea 153) cada bit de *flag* para ver si ha llegado el evento de ese botón.

Notificación de nivel de byte

También es posible enviar un valor entero anulando el valor de la última notificación no leída o no leída y también incrementando el valor anterior, lo cual es muy interesante. También puede usar esta notificación como un semáforo. Todo esto se puede hacer pasando diferentes valores al tercer parámetro (`eAction`):

```
eNoAction
eSetBits
eIncrement
eSetValueWithOverwrite
eSetValueWithoutOverwrite
```

Y para el caso del semáforo (binario) como las diferentes funciones `xTaskNotifyGive() ulTaskNotifyTake()` .

Anulación e incremento del valor de notificación

Expandamos el código de la lista anterior agregando una segunda *task* de menor prioridad que verificará cada 5 segundos si hay notificaciones para ello. Esta notificación se enviará desde la *task* existente cada vez que

se reciba un evento de interrupción externa 0 (por el mecanismo de notificación que ya se ha implementado).

Trabajaremos con el envío de notificaciones utilizando el parámetro `eAction` con el valor `eSetValueWithOverwrite` , para comenzar con la notificación del contador 0 y el valor `eIncrement` (con cada recibo de evento) para permitir aumentar el valor de la notificación, que será leída por la otra *task* cada 5 segundos más el tiempo para que la *task* que maneja las interrupciones externas esté en espera y brinde la oportunidad de ejecutar la segunda *task* de menor prioridad.

El uso de una contranotificación permite que no se reciban todas las notificaciones, pero cuando la recibe puede saber cuántos eventos han ocurrido. ¿Alguna vez ha notado que su t *smart tv* (según el fabricante, imagino) a veces no responde al comando del control remoto para bajar el volumen que ha encendido varias veces en el momento exacto que presiona, pero después de un tiempo? Bueno, debe estar haciendo algo más prioritario que eso, pero la parte de interrupción tomó los comandos y puede haber aumentado o disminuido un valor de volumen que se leyó en algún momento. [7] .

Los siguientes serán los cambios sugeridos para esta implementación y la fuente estará disponible en *github* en [8] : https://github.com/maxback/multitarefa_na_pratica/tree/master/NotificacaoTasksDoisLEDsContadorExt0

Cree una *task* con menor prioridad para recibir notificaciones guardando su identificador para enviar notificaciones:

```
xTaskCreate(taskContadorExt0Func, "gxTaskContadorExt0",
128, NULL, 1, &gxTaskContadorExt0 );
```

En la función de *task* existente, enviar una primera notificación al valor 0 y luego a cada notificación de ext0 envía un incremento:

```
if( gxTaskContadorExt0 )
{
xTaskNotify(gxTaskContadorExt0, 0,  eSetValueWithOverwrite);
}
//…
```

```
    // si el evento ext 0 ve envía una notificación a otra
tarea
    if( (i == 0) && (gxTaskContadorExt0) )
    {
    xTaskNotify(gxTaskContadorExt0, 0, eIncrement);
    }
```

Ahora, el recibo, dentro del *loop* de la nueva *task* , verificará cada 5 segundos si tiene una notificación, enviándola por serie. (sigue la función completa):

```
    void taskContadorExt0Func( void *pvParameters )
    {
    uint32_t ulLeituras;
    char szMsgContador[100];

    for (;;)
    {
    // Comprueba la notificación sin esperar
    if( xTaskNotifyWait(0, 0, &ulLeituras, 0) == pdPASS)
    {
     // imprimir contador
    sprintf(szMsgContador, "Contador: %d", ulLeituras);

    SerialDebugComSemaforo(szMsgContador);

    };

    //espera 5 segundos
    vTaskDelay(5000 / portTICK_PERIOD_MS);
    }
    }
```

Un detalle que no explicamos, y notará si inspecciona cuidadosamente el código, es que ahora usamos un semáforo y una función para enviar datos a través de la serie, para evitar los conflictos de acceso a los recursos ya discutidos en el Capítulo 4 - Trabajando con semáforos .

Capítulo 7 - Tiempo de software (software timming)

Pensemos en la estrategia de implementar el *loop* de función `main()`, que usamos (y todavía hay situaciones para usarlo) cuando no teníamos un sistema operativo a nuestra disposición. Era común tener variables de control de *loop* global, como *timeouts* de espera que se redujeron en las interrupciones del *timer* y se monitorearon en el *loop* principal (o incluso la interrupción en sí) si era demasiado crítico esperar su prueba en el bucle principal.

Fragmentos de código dispersos entre la interrupción y el *loop* principal, así como cualquier función. Si el sistema incrustado comenzara a crecer, tendríamos que agregar más variables distribuyendo este monitoreo en varios puntos de los archivos fuente.

FreeRTOS y RTOS generalmente proporcionan mecanismos que simplifican la tarea de realizar una función periódicamente o solo una vez después de un *timeout* de espera. Permitir el control abstracto de una variable global y los problemas críticos de la sección que podrían ocurrir, causando corrupción en la ejecución del sistema.

FreeRTOS activa y desactiva algunas de sus funciones en tiempo de compilación. Dado que el *timer* de fondo es una *task* con la que nos comunicamos a través de las funciones de la API, debemos verificar que el temporizador lo active con el valor 1 en el archivo FreeRTOSConfig.h en la carpeta *lib* correspondiente a freeRTOS en la estructura arduino.

```
#define configUSE_TIMERS 1
```

Si no tiene este valor, puede cambiarlo, o incluso copiar el archivo a la carpeta de su proyecto y poner el nuevo valor solo en esta copia, para establecer esto solo para el proyecto actual y no para todos.

Ejecución periódica de una función

Hasta ahora usamos semáforos o colas para comunicar eventos y datos entre *tasks* o

Veamos un ejemplo de una función que se ejecutará periódicamente, como se configuró sin manipular ninguna variable, solo ejecutando el código deseado.

A diferencia de la función asociada a la *task* , no tiene un *loop* infinito (y no debería tener) y no tiene que recargar ningún valor asociado con el control de ejecución.

Listagem 12:
BlinkELEDTimerFunc

```
001: #include <Arduino_FreeRTOS.h>
002: #include <timers.h>
003:
004: void TaskAnalogRead( void *pvParameters );
005:
006: // función de temporizador
007: void callbackBlinkLED(TimerHandle_t xTimer);
008:
009: void setup() {
010:    // Identificador del temporizador
011:   TimerHandle_t gxTimer;
012:
013:   pinMode(LED_BUILTIN, OUTPUT);
014:
015:    // initialize serial communication at 9600 bits per
second:
016:   Serial.begin(9600);
017:   while (!Serial);
018:
019:   xTaskCreate(TaskAnalogRead, (const portCHAR *)
"AnalogRead",
020:     128, NULL, 1, NULL);
021:
022: // crear temporizador
023: gxTimer = xTimerCreate("TimerBlinkLED",
024: 1000 / portTICK_PERIOD_MS, // llamar a func. 1 en 1s
025: pdTRUE,    //// con autoreload (periódica)
026: 0, // ID del temporizador 0 que indica el índice de parpadeo
de los LED al inicio
027: callbackBlinkLED );
028:
029:   if(gxTimer != NULL)
030:     xTimerStart ( gxTimer, 0);
031: }
032:
033: void loop()
034: {
035:    //
```

```
036: }
037:
038: void callbackBlinkLED(TimerHandle_t xTimer )
039: {
040:     /* Varios patrones de parpadeo basados en texto:
041:      * L - Encendido 9 por un segundo período
042:      * D - Led apagado por un segundo período
043:      * Al llegar al final del último carácter, repite el
patrón.
044:      * Se establecen 4 estándares diferentes.
045:      */
046:     // acceso por 1 segundo y apagado por 4 (2x), encendido
por 1 segundo y apagado por 9 (2x),
047:     // acceso por 5 segundos y apagado por 5 y apagado por 1
048undo y encendido 4 (2x)
048:     // (contrario al primero)
049:    char normas[] =
"LDDDDLDDDD_LDDDDDDDDDLDDDDDDDDD_LLLLLDDDDDLLLLLDDDDD_DLLLLDLLL
L";
050:
051:    uint32_t nIndice;
052:     // Timer Led ID el valor del índice
053:    nIndice = ( uint32_t ) pvTimerGetTimerID( xTimer );
054:
055:     // decide según el carácter del pádel
056:    char nivel = normas[nIndice] == 'L' ? HIGH : LOW;
057:
058:     // si '_' se mantiene actualizado
059:    if(normas[nIndice] != '_')
060:    {
061:      digitalWrite(LED_BUILTIN, nivel);
062:    }
063:
064:    nIndice++;
065:    if(normas[nIndice] == '\0')
066:    {
067:        nIndice = 0;
068:    }
069:
070:     // Guardar en ID de temporizador el valor del índice
```

```
071:    vTimerSetTimerID( xTimer, ( void * ) nIndice );
072: }
073:
074:
075: void TaskAnalogRead(void *pvParameters)
076: {
077:    (void) pvParameters;
078:    for (;;)
079:    {
080:      int sensorValue = analogRead(A0);
081:      Serial.println(sensorValue);
082:      vTaskDelay(1);   // 1 tick delay (15ms)
083:    }
084: }
```

En la línea 23, dentro de la función de configuración, se llama a la función `xTimerCreate` para crear un nuevo temporizador (*timer*). El primer parámetro solo proporciona un nombre legible para el temporizador (ya que las *tasks* pueden tener múltiples *timers* con la misma función). En el segundo parámetro, se pasa un valor en *ticks* para determinar el período en el que se llamará a la función del *timer* en la línea 24. En nuestro caso, se especificó 1 segundo (`1000 / portTICK_PERIOD_MS`).

El tercer parámetro indica si la función será periódica o no. En nuestro caso pasamos el valor `pdTRUE` .

El siguiente parámetro, la ID del temporizador (*timer ID*), es un valor de 32 bits de propósito general que puede canjearse dentro de la función. Comencemos con el valor 0, que indica que se debe ejecutar el primer elemento del patrón de *blink* del LED (lo veremos en la función del *timer*).

Todo lo que queda es pasar la función del *timer* en la línea 27. La función devuelve el identificador del *timer* en la línea 23, que se guarda en `gxTimer` , para usar en la función `xTimerStart()` , que comenzará a ejecutarse inmediatamente en la línea 30.

En la función de temporizador `callbackBlinkLED()` , en la línea 38 tenemos una matriz (*array*) de caracteres con los caracteres 'L', 'D' y '_' que indican los estados del LED en cada llamada de función. ‘L’ para encendido (Salida lógica 1 al encender el LED), ‘D’ para apagado (salida lógica 0, rompiendo la corriente en el LED) y para _ ’para mantenerlo como está (en la práctica, separa diferentes patrones en la cadena).

La función comienza copiando la ID del *timer* en la variable `nIndice` , llamando a la función `pvTimerGetTimerID` con el identificador de *timer* actual, recibido por parámetro en la función de *timer* . Recuerde que pasamos el valor 0 al crear el o.

Este valor se utiliza como base para encender (o apagar) el LED y se cambia al siguiente índice hasta que alcanza el último, cuando vuelve al valor 0.

Un detalle importante

Para que una segunda ejecución de la función del *timer* acceda a un índice que no sea la *string (* cadena de patrón del LED), el valor de la variable de índice debe almacenarse en la ID del *timer* para leerse más tarde, a medida que finaliza la función y sus valores de variable las ubicaciones se pierden (a diferencia de una función de *task* , que está en un bucle infinito). Esto se hace en la línea 71, donde el nuevo valor de índice se guarda en el *timer* actual utilizando la función `vTimerSetTimerID()` , pasando el identificador del temporizador y el valor (con un *type* a `void *` , que es el tipo esperado). por función).

Ejecución única de una función.

Podemos crear un temporizador de ejecución única o disparo único pasando el valor falso `pdFALSE` al crearlo:

```
gxTimerUnico = xTimerCreate("TimerUnico",
10000 / portTICK_PERIOD_MS,
pdFALSE,
0,
callbackTimerUnico );
```

¿Y después?

Bueno, entonces espera un momento en el que desea configurar el temporizador, como presionar un botón, y llama a la función `xTimerStart()` .

Si se completa el tiempo establecido (como 10 segundos en el ejemplo), la función establecida se llama una vez y luego no más. Se requerirá una segunda llamada a `xTimerStart()` para un nuevo

temporizador. Si desea volver a cargar el temporizador que ya está en progreso, simplemente llame a `xTimerStart()` , que se reiniciará en 10 segundos (por ejemplo).

Sin embargo, si ocurre algún evento y desea cancelar el conteo de tiempo, simplemente llame a la función `xTimerStop()` antes de que se agote el tiempo.

Segunda parte: diseño de captura de lectura de giroscopio

Para permitir un enfoque un poco más práctico, mejor a la altura del título de este libro, presentaremos el diseño de un colector de lectura giroscópica.

Este proyecto implementa la lectura de un sensor giroscópico utilizando I2C y el envío de estas lecturas a un servidor a través de solicitudes HTTP GET a través de la conexión de red WiFii.

Este proyecto se basa en el proyecto TCC que ejecuté, que está disponible en la siguiente dirección web:

http://www.riuni.unisul.br/handle/12345/5249

En ese proyecto, primero implementamos esta aplicación en otro hardware y luego la trasladamos a Arduino, en función del uso de freeRTOS en ambos proyectos, con el objetivo de reutilizar la mayor cantidad de código posible. [10] .

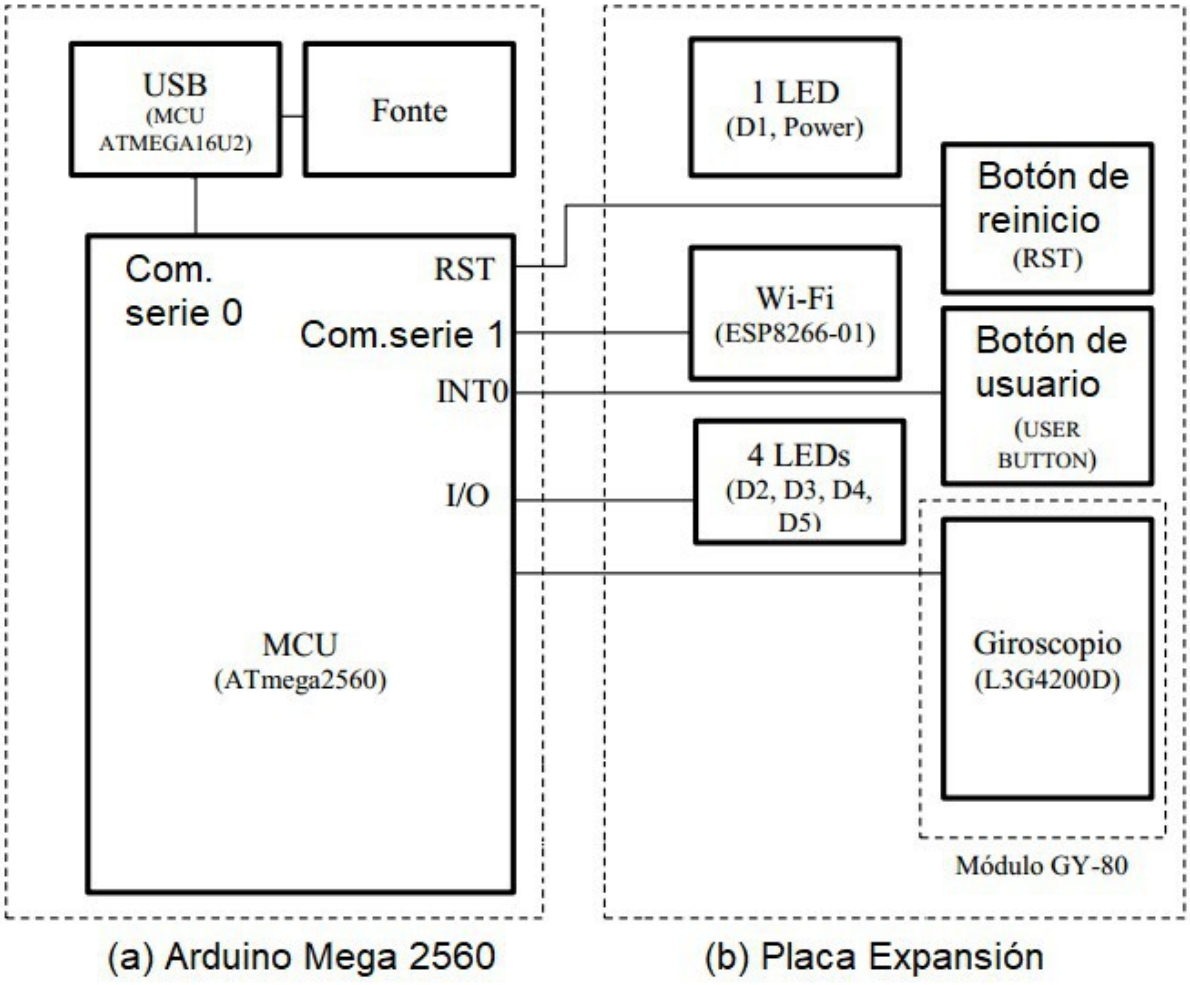
¿Qué se presentará?

Presentaremos los circuitos básicos utilizados, el modelo de módulo y luego discutiremos los detalles de implementación, incluido el ajuste del archivo de asignación de memoria, para adoptar otro modelo de asignación sugerido por freeRTOS.

Esta vez se desarrollará un ejemplo más grande módulo por módulo, en el antiguo enfoque de arriba hacia abajo(*TOP-DOWN*), es decir, desde una descripción más general, con el diagrama del módulo a los detalles de cada módulo.

Capítulo 8 - Hardware del proyecto

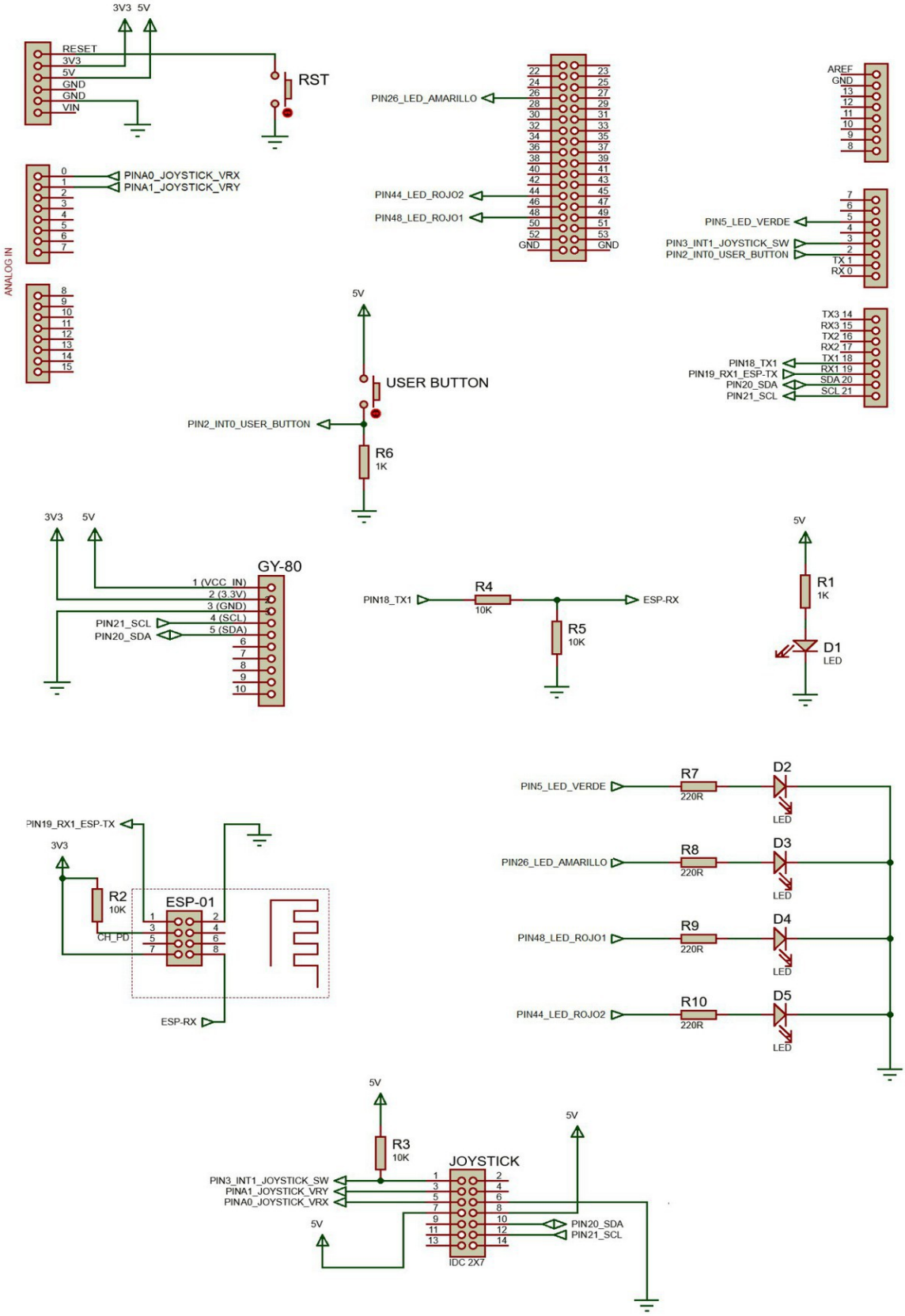
El siguiente es el diagrama de bloques de hardware de este proyecto. En (a) tenemos el bloque Arduino Mega 2560, su USB incorporado y una fuente externa compatible con su entrada de alimentación. Ya en (b) tenemos los bloques en una placa de expansión, que tiene 1 LED de alimentación, simplemente conectado a la alimentación Vcc proporcionada por el propio Arduino (a través de una resistencia); Un botón de reinicio (conectado a la propia señal de reinicio de Arduino). Además de estas características accesorias, que no hacen más que replicar lo que ya está en la placa Arduino, también tenemos un botón de propósito general (Botón de usuario), 4 LED indicadores, un módulo de sensor (donde está contenido el sensor giroscópico) y un Módulo wifi ESP8266-01.



Esta placa de expansión se puede montar en una placa de circuito universal agregando pines ubicados en lugares convenientes para la conexión Arduino Mega requerida.

Diseño de placa de expansión

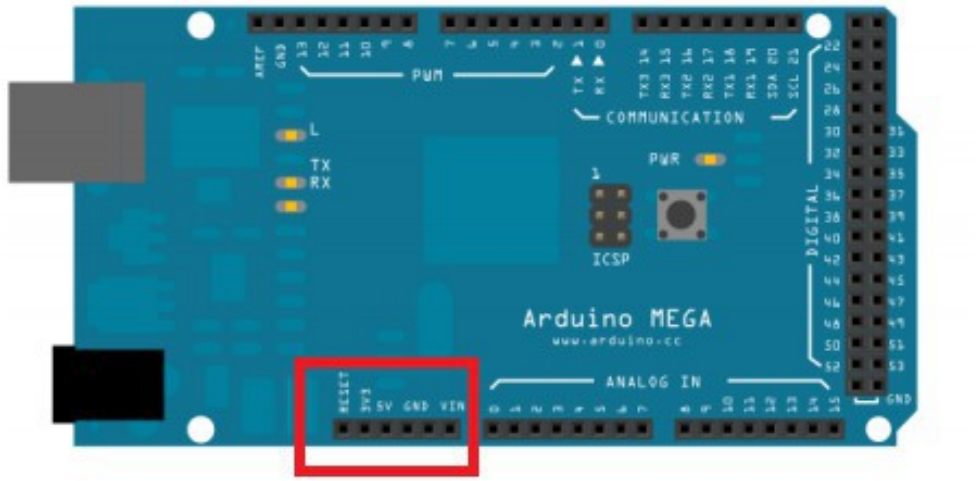
El siguiente es un diagrama esquemático de esta placa de expansión que se explicará parte por parte a continuación.

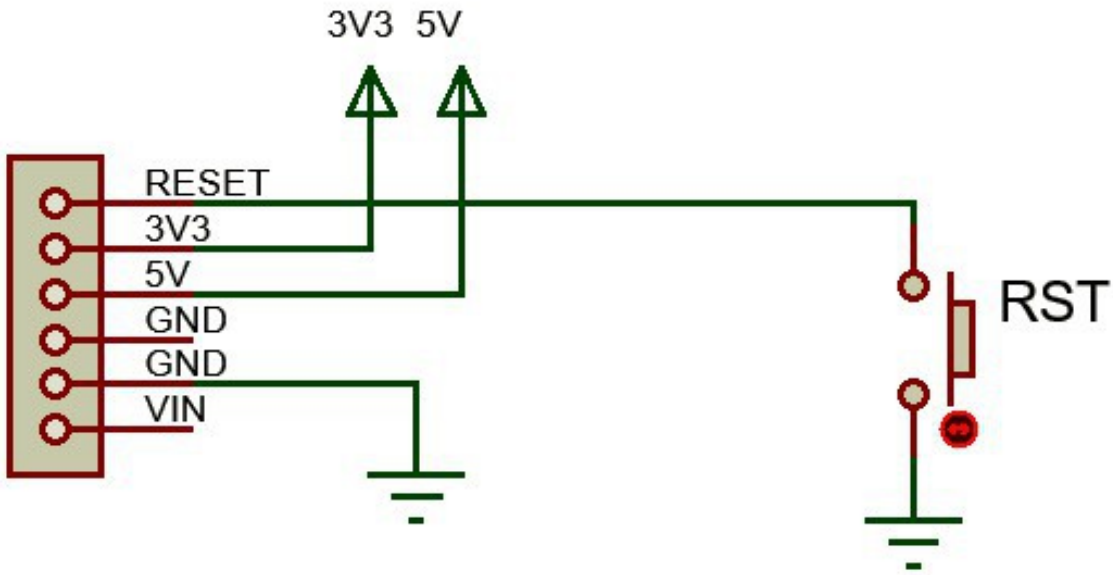

3V3
5V
RESET
3V3
5V
GND
GND
VIN
RST
ANALOG IN
PINA0_JOYSTICK_VRX
PINA1_JOYSTICK_VRY
PIN26_LED_AMARILLO
PIN44_LED_ROJO2
PIN48_LED_ROJO1
GND
AREF
GND
PIN5_LED_VERDE
PIN3_INT1_JOYSTICK_SW
PIN2_INT0_USER_BUTTON
TX 1
RX 0
TX3 14
RX3 15
TX2 16
RX2 17
TX1 18
RX1 19
SDA 20
SCL 21
PIN18_TX1
PIN19_RX1_ESP-TX
PIN20_SDA
PIN21_SCL
5V
USER BUTTON
PIN2_INT0_USER_BUTTON
R6
1K
3V3
5V
GY-80
1 (VCC_IN)
2 (3.3V)
3 (GND)
4 (SCL)
5 (SDA)
PIN21_SCL
PIN20_SDA
PIN18_TX1
R4
10K
ESP-RX
R5
10K
5V
R1
1K
D1
LED
PIN19_RX1_ESP-TX
3V3
R2
10K
ESP-01
CH_PD
ESP-RX
PIN5_LED_VERDE
R7
220R
D2
LED
PIN26_LED_AMARILLO
R8
220R
D3
LED
PIN48_LED_ROJO1
R9
220R
D4
LED
PIN44_LED_ROJO2
R10
220R
D5
LED
5V
R3
10K
JOYSTICK
PIN3_INT1_JOYSTICK_SW
PINA1_JOYSTICK_VRY
PINA0_JOYSTICK_VRX
5V
5V
PIN20_SDA
PIN21_SCL
IDC 2X7

Comentemos este diagrama esquemático en partes. Cada parte también contendrá el diseño de un arduino resaltando los conectores donde se conecta esa parte de la placa de gastos. Si elige cablear los circuitos en un *protoboard* o algo así, esta característica puede facilitar la colocación de los pines, donde también se indica la numeración.

Botón de reinicio (reset)

Conectado al pin RESET de arduino, permite reiniciar el circuito cuando la tarjeta de expansión cubre la tarjeta base. Es un botón pulsador (*push-botton*) que conecta a tierra este pin (GND).



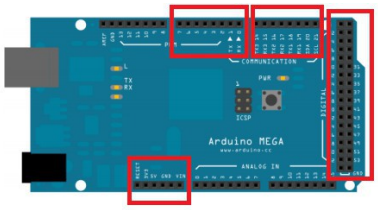


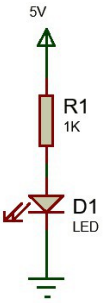
LED de power e LEDs indicadores

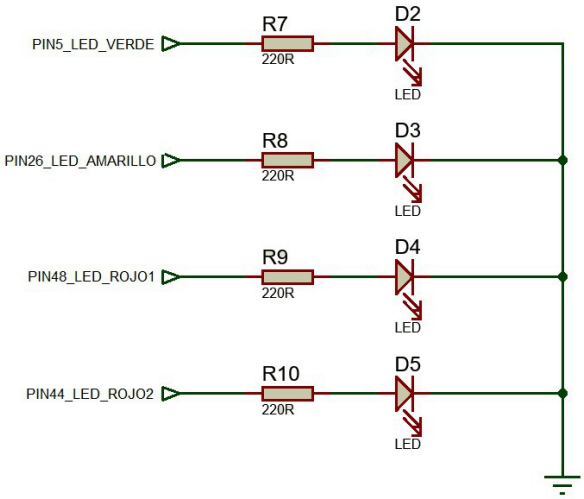
En el mismo conector que el botón de reinicio, un LED (D1) y una resistencia de 1k (R1) forman el circuito para informarle si ha llegado energía a la placa. Si no monta una placa, este subcircuito también es superfluo (como lo es el botón de reinicio).

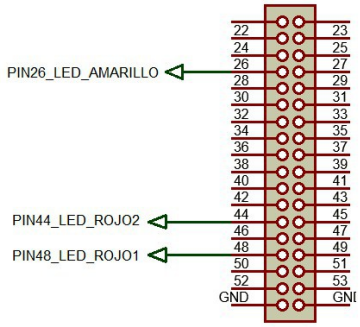
Los LED D2, D3, D4 y D5 conectados a los pines 5, 26, 18 y 44 tienen el objetivo de permitir el diagnóstico del funcionamiento del sistema. Estos LED se utilizan en nuestro proyecto para indicar el estado de la

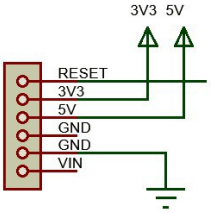
comunicación y, en caso de pila de memoria, estoy mostrando qué *task* fue responsable del problema. Puede elegir los LED de su elección de colores y tamaños, teniendo cuidado de diseñar una resistencia adecuada, teniendo en cuenta las características del puerto arduino y el tipo de LED.

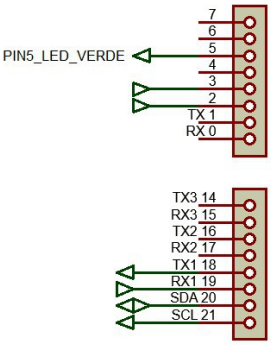








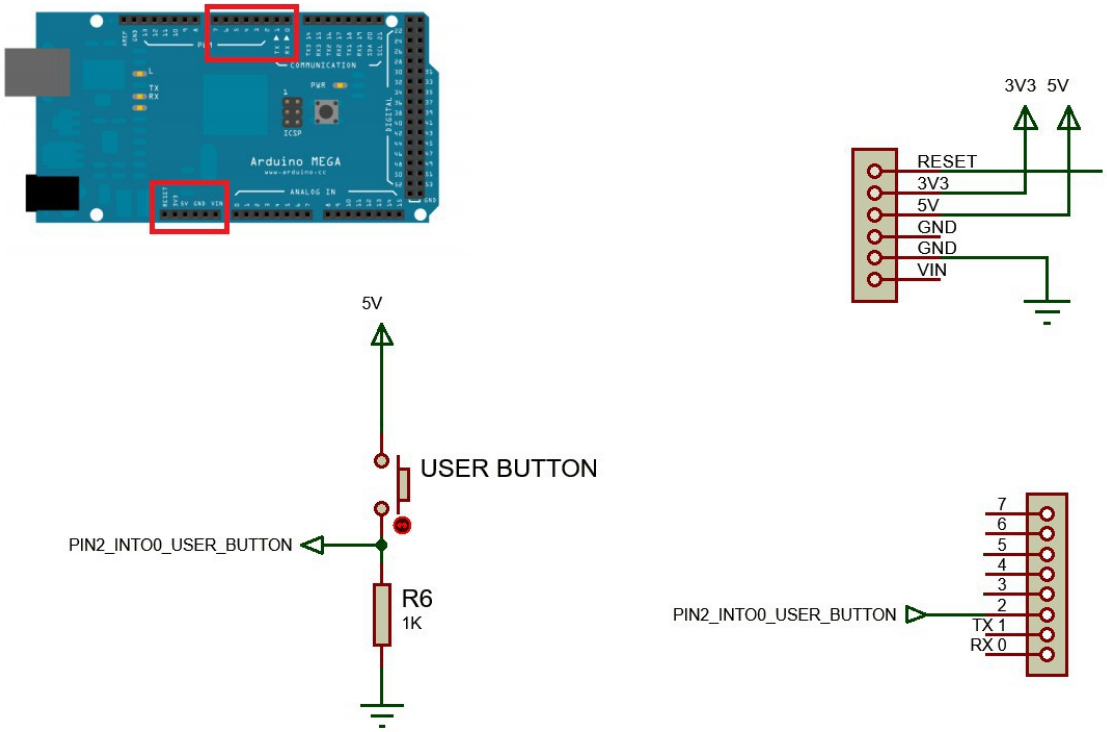




Botón de usuario

Es un botón de propósito general que en esta aplicación captura el evento de su presión informada al host (a través de http) y también se utiliza para ajustar el filtro de entrada del giroscopio.

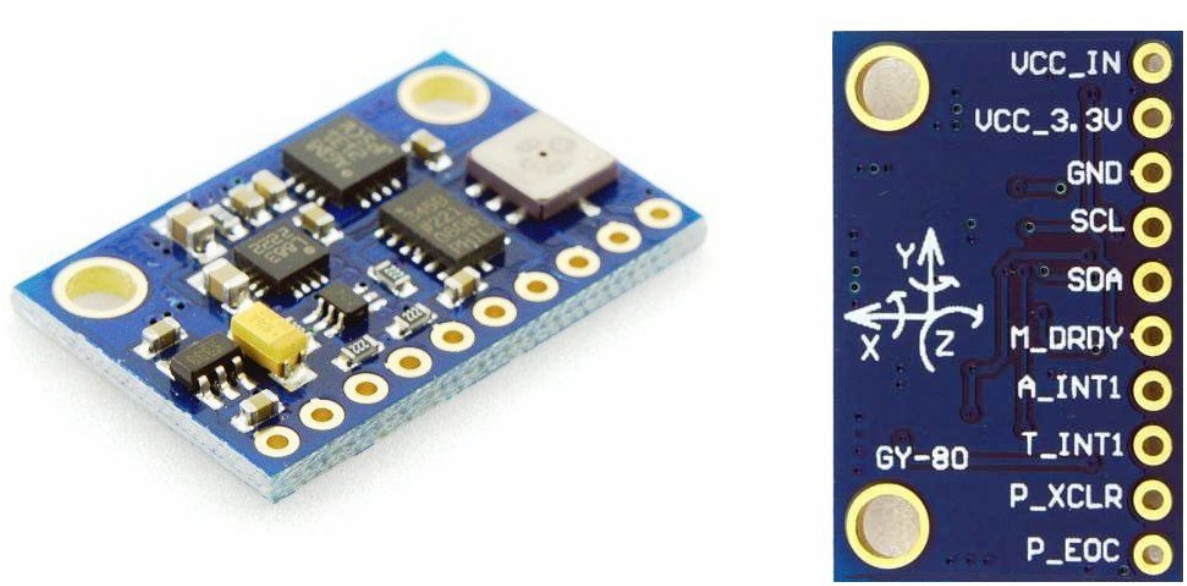
Es un simple botón pulsador *push-botton* y una resistencia *pull-down* de 1k (R6) para garantizar el nivel lógico 0 cuando se activa la entrada 0 si no se presiona el botón.



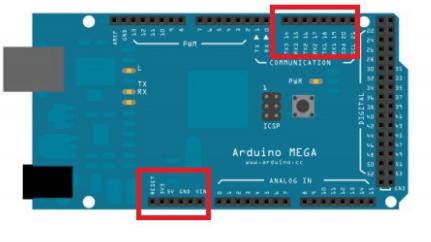
Está conectado al pin 2, donde se captura la interrupción externa 0 (INT0).

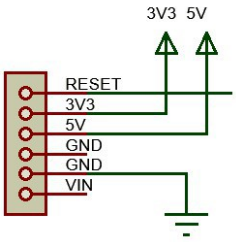
Giroscopio

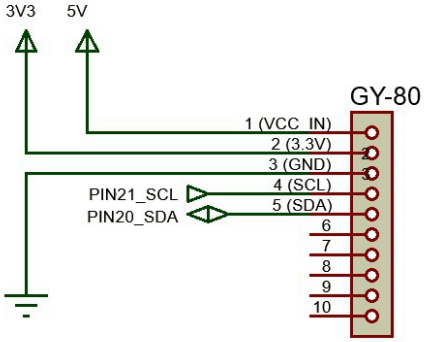
En este proyecto utilizamos un módulo sensor GY-80 que teníamos a nuestra disposición. Incluye varios sensores [11] en una placa pequeña con comunicación I2C.

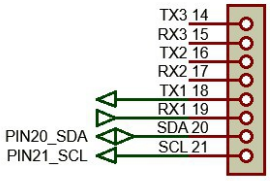


En nuestro esquema, simbolizamos este módulo simplemente como un conector de 10 pines, indicando el significado de cada módulo, ya que está impreso en la PCB que acabamos de ver. Tenga en cuenta que solo se han realizado las conexiones necesarias para la comunicación I2C con el módulo sin interrupciones externas.





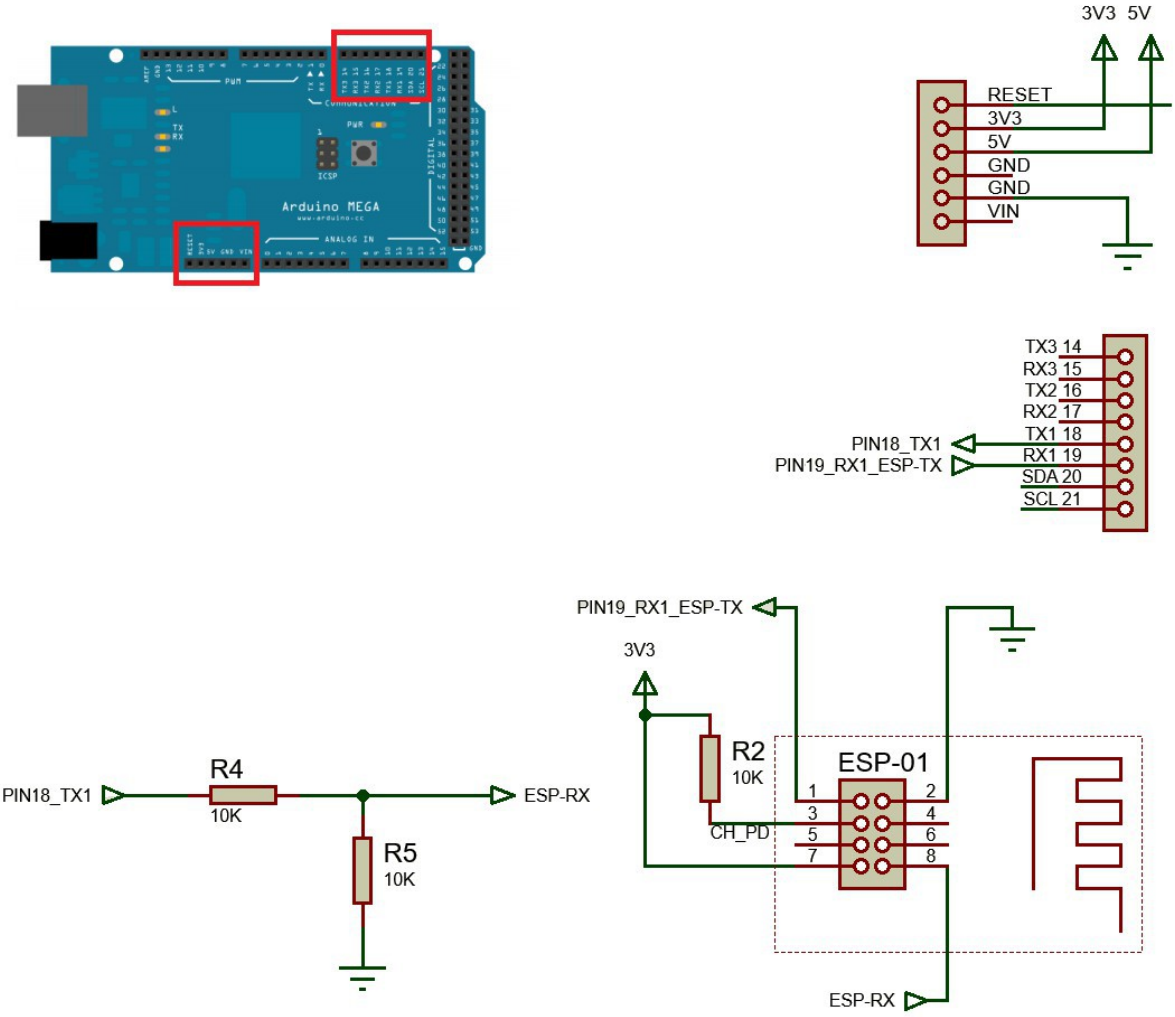




El giroscopio IC es el STMicroeletronics L3G4200D.

Módulo de Wi-Fi ESP

Este proyecto utilizó un módulo Wi-Fi ESP8266-01 conectado a través del puerto de comunicación serial 1 a Arduino.



Dado que este módulo funciona con 3.3V y Arduino con 5V, las señales de Arduino tienen un alto nivel lógico con 5V, y para el módulo debe estar alrededor de 3.3V. Para adaptar este nivel (en la salida de transmisión Arduino Tx) se ha agregado un divisor de voltaje formado por las resistencias R4 y R5 que entregan un nivel lógico apropiado al pin receptor (Rx) del módulo.

La señal Tx del módulo no necesita ser adaptada para ser entendida por el arduino Rx, ya que para los valores estándar TTL alrededor de 3V

todavía se consideran como un alto nivel lógico [12] .

Las conexiones en los pines 3 y 7 del módulo están configuradas para restablecer el módulo solo cuando el equipo está encendido y mantenerlo en el módulo de ejecución (en su lugar y en modo de actualización) todo el tiempo.

En mi prototipo, elegí soldar el módulo directamente a la placa base, ya que tenía la intención de mover mucho el equipo, pero puede usar dos barras hembra de 4 posiciones, flanqueadas para permitirle conectar el módulo a estas barras pin.

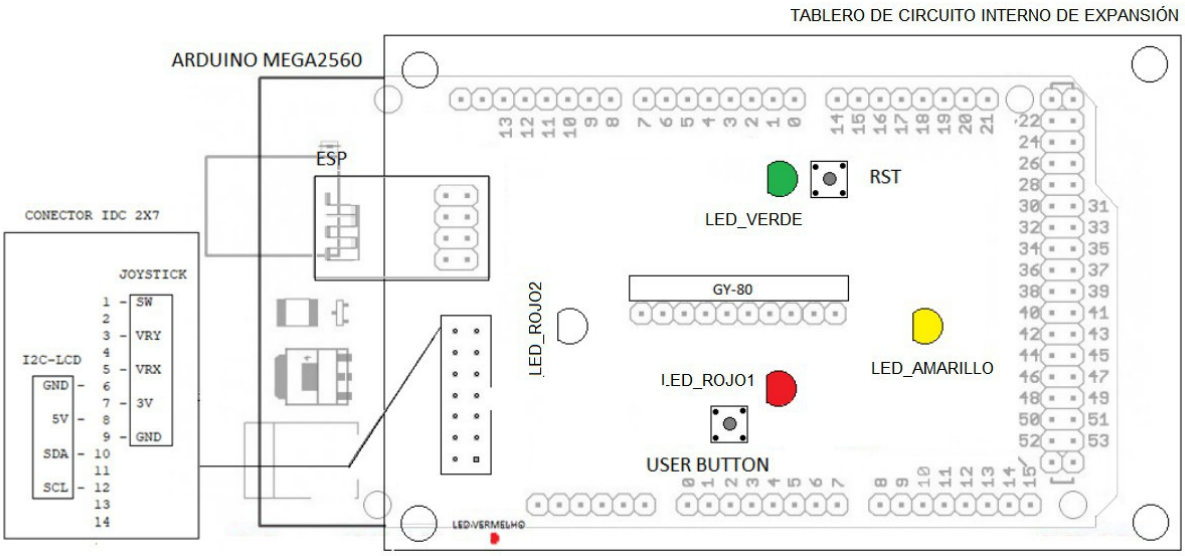
Este módulo se ordena como un módulo esclavo, a través de comandos AT enviados y la interpretación de cuyas respuestas también son leídas por el serial. Este módulo generalmente se envía con software integrado estándar (ya que puede programarse directamente) con soporte de comando AT.

Conector de expansión (Joystick)

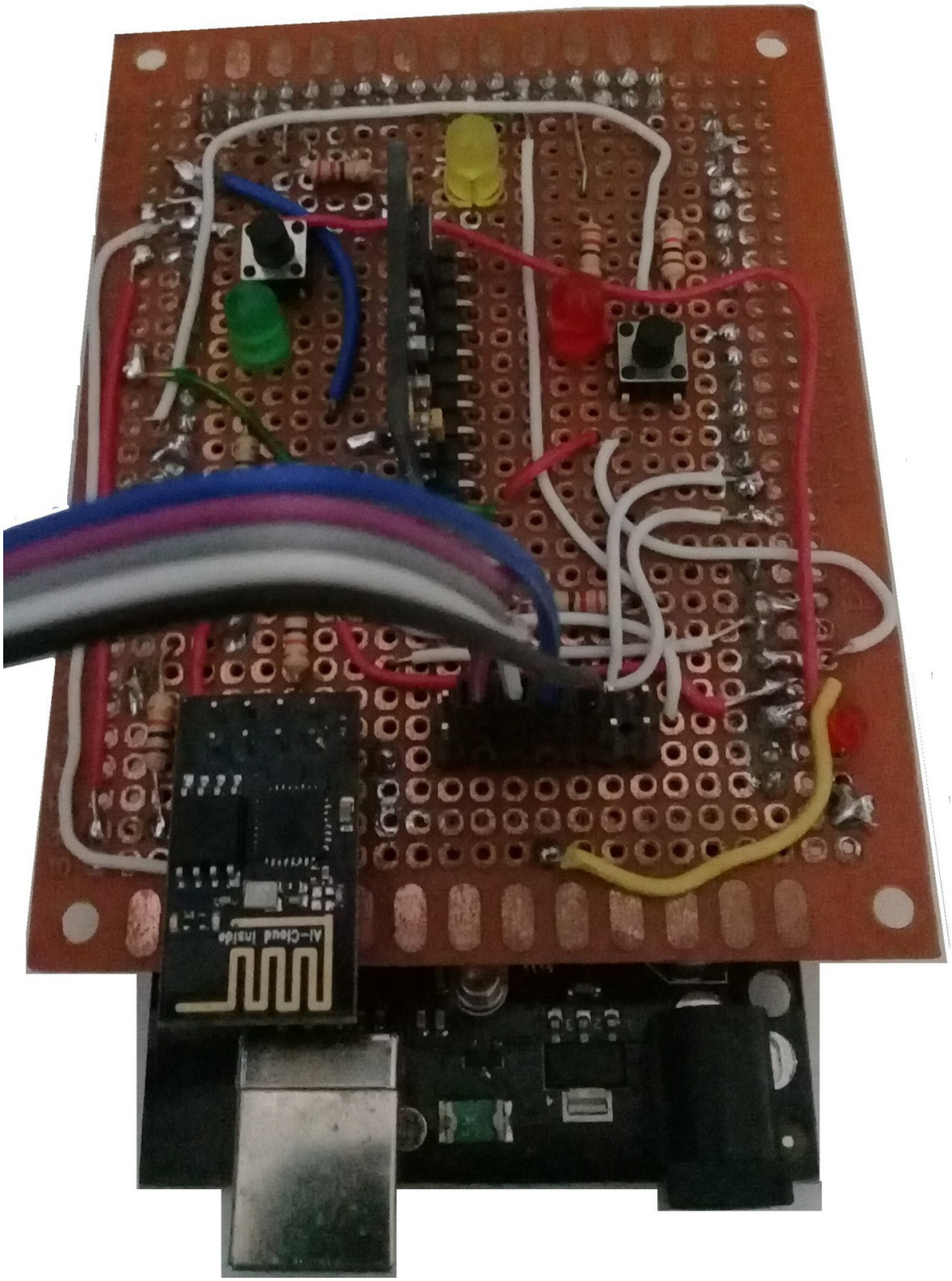
En el esquema aparece un conector llamado joystick que desempeña el papel de punto de expansión para periféricos, como un joystick formado por indicadores de puerta y un interruptor, así como un canal I2C. No utilizaremos este conector en el proyecto, pero se propondrá más adelante como ejercicio para su uso. Si lo prefiere, puede ir al Apéndice C - Joystick y conector de expansión de canal I2C para obtener una descripción de cómo se puede usar.

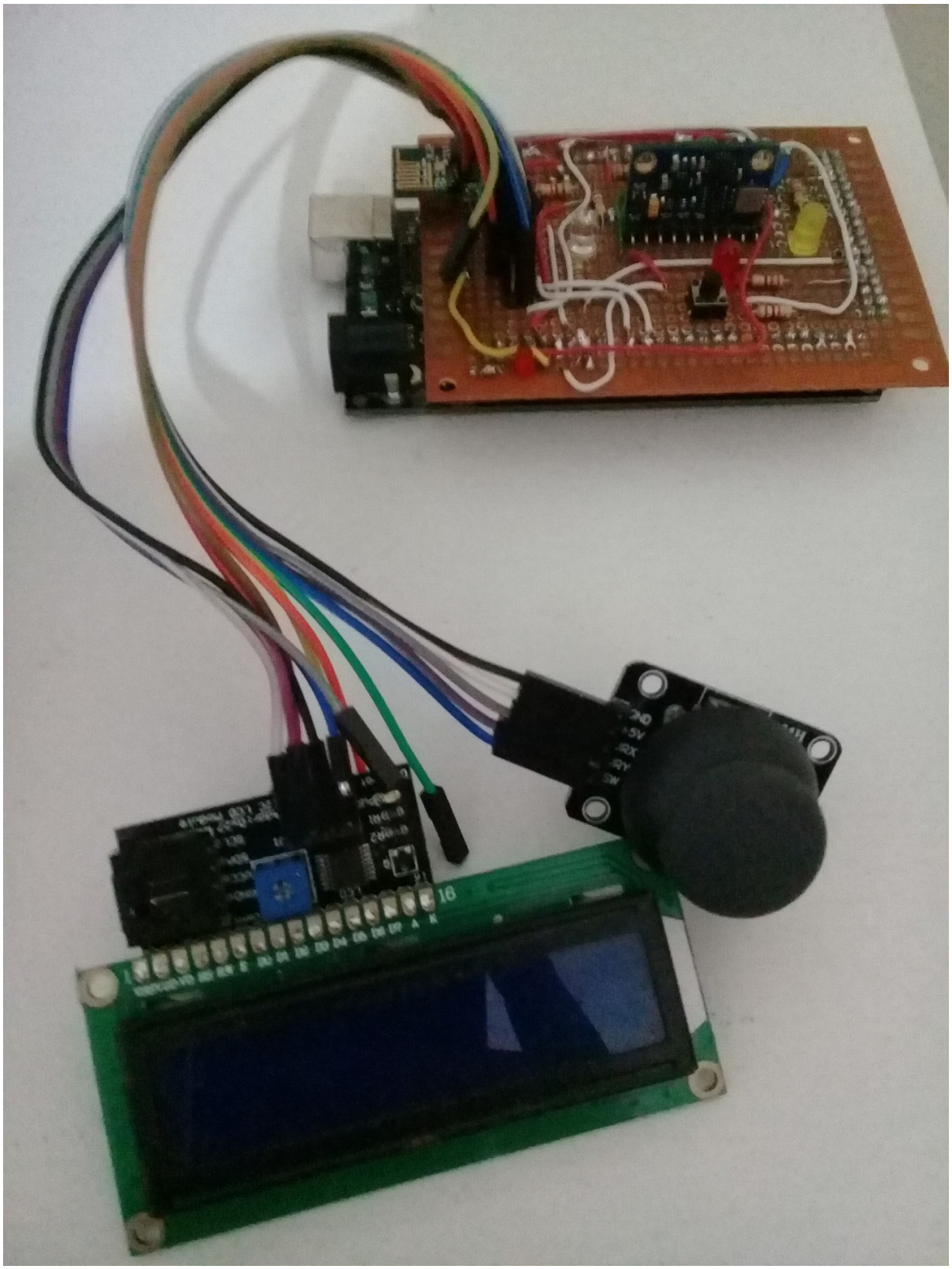
Montaje de tarjeta de expansión

El montaje puede variar mucho según las decisiones de colocación de componentes. El siguiente es un esquema mecánico sugerido, que coloca la placa del sensor GY-80 perpendicular a la placa universal en el centro de la misma, rodeada por los LED indicadores. La tarjeta del módulo wifi y el conector del joystick están en el borde opuesto que coincide con el lado Arduino Mega donde están los conectores USB y de alimentación externa.



Las siguientes son fotos de mi prototipo, con un *joystick* y una *display* LCD con una interfaz I2C (que es una opción para expandir el proyecto).

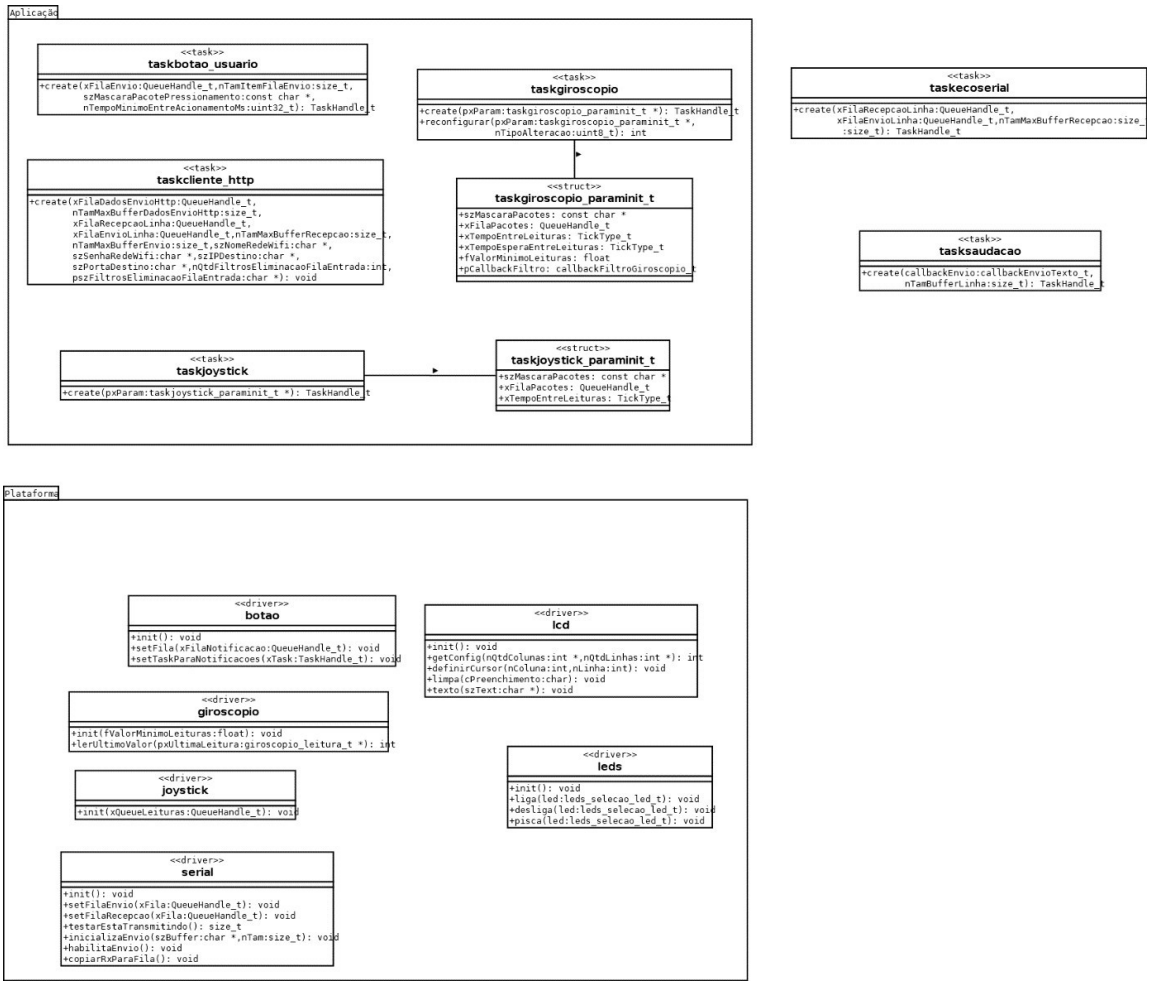

AI-Cloud Inside



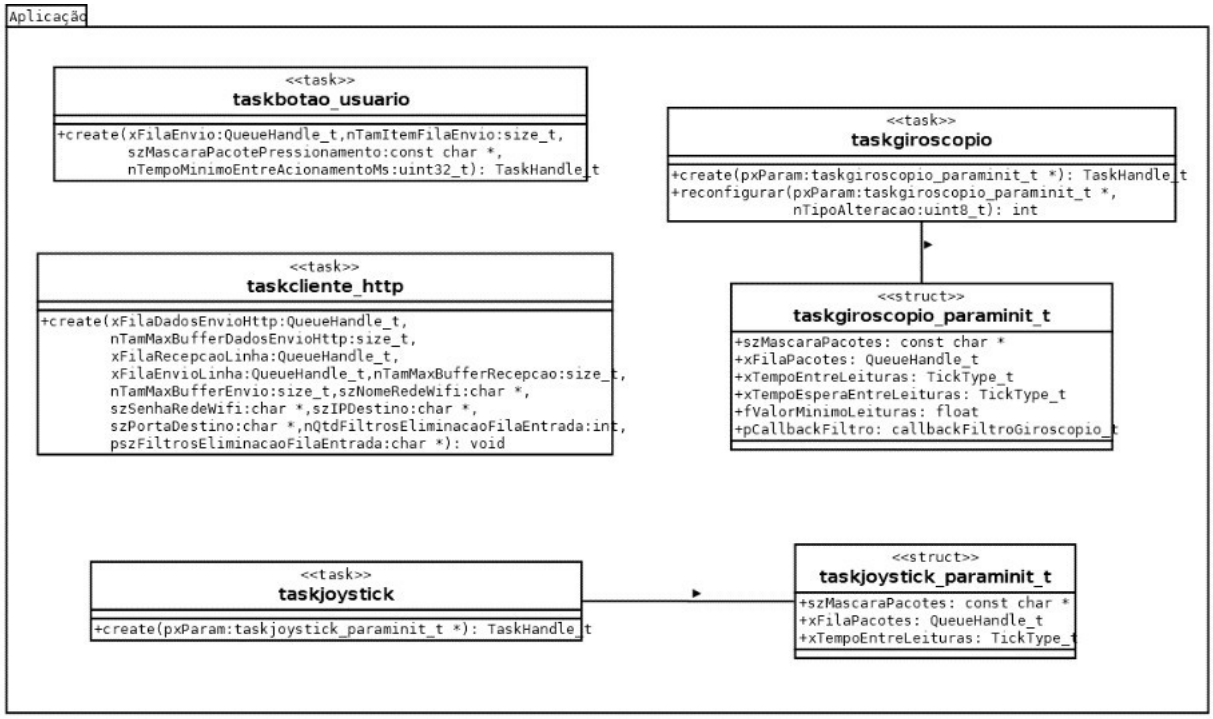
Capítulo 9 - Organización del software del proyecto

Para este proyecto, adoptaremos una mayor organización de tareas y módulos, ya que de hecho debe hacerse en un proyecto realmente serio y profesional (al menos los que desea después del mantenimiento, con modificaciones, expansiones y adaptaciones a las nuevas configuraciones de hardware) .

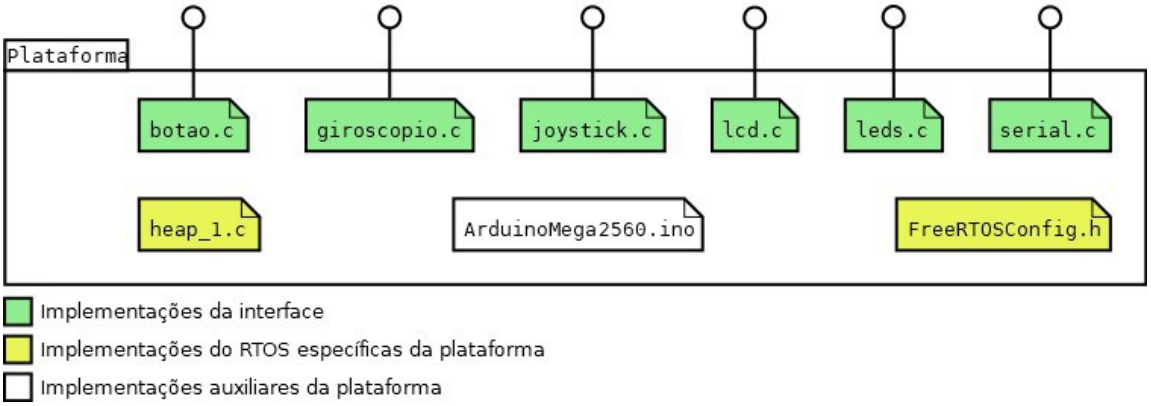
La implementación completa se realizó utilizando solo C y no C ++. Sin embargo, esto no impide que se use el modelado UML, por ejemplo, ver los módulos como objetos. De esta forma, podemos visualizar la organización del *softwarwe* integrado de este proyecto a través de diagramas de clases, nivel de aplicación (y servicios) y separación de paquetes de plataforma (incluido soporte de *hardware* de circuito).



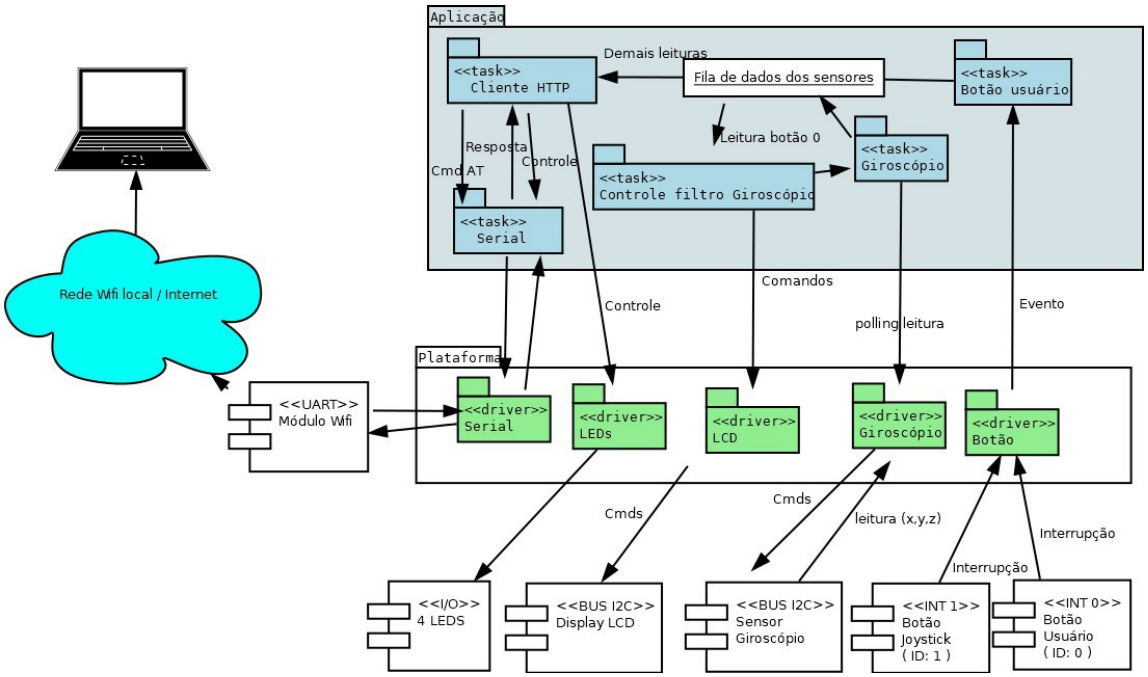
Bueno, este es el panorama general. Para facilitar su examen, mostraremos a continuación dividido en los paquetes de aplicación y plataforma (dejando de lado *taskecoserial* y *tasksaldacao* , ya que sus propuestas se usan solo para pruebas):



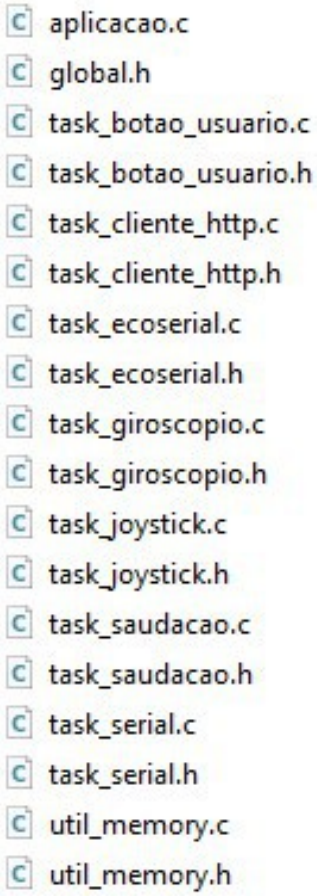
La siguiente es una descripción general de los archivos y *drivers* de hardware específicos de arduino:



Además de estos archivos, tenemos las fuentes de producción que se conectan a ellos, lo que resulta en la siguiente plataforma completa:

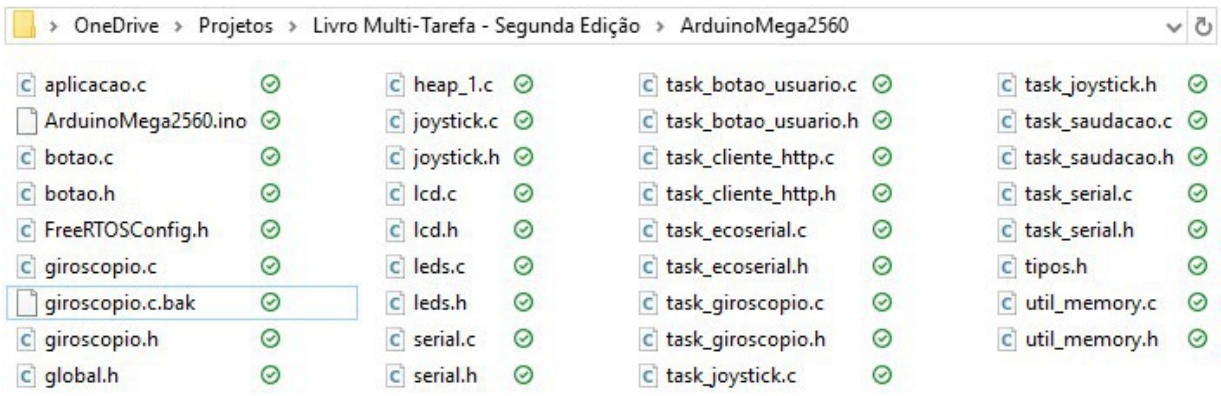


Esta parte de la aplicación se implementa en los siguientes archivos:



Como puede ver, este proyecto tiene varios archivos, ya que separa el proyecto en varios archivos, que funcionan en conjunto con el archivo Arduino llamado ArduinoMega2560.ino.

El resultado final se muestra en la siguiente imagen en la carpeta de fuentes:



Todas estas fuentes se enumeran aproximadamente para su copia en el Apéndice D - Código fuente del proyecto práctico (segunda parte) . A partir de ahora examinaremos ciertas partes del código, que siempre se indicarán en un sub-elemento del archivo adjunto y al que se puede acceder fácilmente navegando por el documento.

Capítulo 10 - Examen de módulos

La comunicación en este proyecto se logra a través del módulo ESP a través de una red inalámbrica con un script PHP (que se puede ver en el Apêndice E – Código do código PHP). A este módulo se accede por la segunda serie de Arduino Mega (Serie1) mediante el envío de comandos AT al módulo. Primero inspeccionaremos a la parte responsable de la comunicación en serie (independientemente del protocolo y el propósito) y luego la implementación real de la comunicación HTTP, que es la guinda del pastel.

Más adelante veremos los otros módulos, que siguen un patrón similar y, por lo tanto, su inspección cuidadosa no agregaría mucho. En un proyecto tan grande, como es el caso, sería un gran dolor dedicar el mismo nivel de profundidad a todos los puntos.

Ruta de ejecución en el módulo de comunicación serial

La primera capa de abstracción se encuentra en el archivo serial.c , que accede a un conjunto de funciones que necesitaba poner en el archivo ArduinoMega2560.ino para llamar desde este módulo, que realiza la gestión de la comunicación orientada a paquetes [13] :

```
    extern void ExecutarSerial1_begin(unsigned long baud);

    extern int ExecutarSerial1_available(void);

    extern  int ExecutarSerial1_read(void);

    extern  void ExecutarSerial1_print(char *s);

    extern int ExecutarSerial1_testarEstahTransmitindo(void);

extern void
ExecutarSerial1_registrarCalbackSerialEvent(callbackGenerico_t
pCallback);
   extern void printlog(int qtdParams, const char *s, int
valor);
```

Aquí está la definición de dos colas freeRTOS:

```
static volatile QueueHandle_t gxColaEnvio = NULL;

static volatile QueueHandle_t gxColaRecepcao = NULL;
```

Estos se definen externamente a través de las funciones `serial_setFilaEnvio()` y `serial_setFilaRecepcao()` , lo que permite definir los puertos de entrada y salida desde fuera del módulo, sin que el módulo tenga que conocer de antemano ni siquiera el propósito y el formato de la comunicación que se producirá. Las colas son caracteres (elemento de tamaño de `char`) y su longitud debe ser definida por el usuario del módulo, para la conveniencia de la comunicación.

Como define `USA_TIMEOUT_SW` está establecido en 1, no se comentarán todas las pruebas de tipo `#if()` . Podría deshabilitarse para las pruebas usando la escritura de vamler en una aplicación de terminal y el manejo del *timeout* de espera sería muy inconveniente.

Dicho esto, tenemos la declaración de una variable de software de *timer* freeRTOS, que usaremos para controlar el tiempo de espera de recepción de paquetes definido en la declaración:

```
static volatile TimerHandle_t gxTimerRecepcao = NULL;
```

Cuyo prototipo de la función de devolución de llamada es:

```
void callbackTimeoutEntreCaracteresRx( TimerHandle_t xTimer
__attribute__((unused)));
```

Creado en `serial_init()` de la siguiente manera (*time* de tiro único para terminar Rx por tiempo de espera):

```
//time de disparo unico para terminar Rx por timeout

gxTimerRecepcao = xTimerCreate("TimerRecepcaoSerial",

  SERIAL_TIMEOUT_RX / portTICK_PERIOD_MS, //converte para
ticks

  pdFALSE, //sem autoreload (one shot)

  0, //timer ID

  callbackTimeoutEntreCaracteresRx );
```

Este *timer* es de una sola vez, es decir, una vez activado, llamará a la función de devolución de llamada (*callback*) cuando haya expirado el *timeout* de espera de recepción y luego no volverá a llamar, a menos que se reactive en una nueva espera de recepción. Sin embargo, si se reciben caracteres, el temporizador se vuelve a cargar y se debe agotar el tiempo de espera nuevamente llamando a su función de inicio: `xTimerStart` (`gxTimerRecepcao, 0`). Adopté una estrategia de hacer esto cada 10 caracteres para evitar muchas llamadas consecutivas.

Si se recibe un delimitador de fin de paquete, o el búfer se llena, este tiempo de espera se deshabilita, llamado `xTimerStop` . Todo esto se puede ver en la función treatCallBackSerialEvent (). [14] :

```
void trataCallBackSerialEvent(void *pvParametos)
{
  (void *) pvParametos;

  char c;
  //acesso exclusivo via semáforo ao buffer
  xSemaphoreTake( SemaforoAcessoBufferRx, portMAX_DELAY );
  while (ExecutarSerial1_available())
  {
   c = (char)ExecutarSerial1_read();
   if(gnTamBufferRx < sizeof(gszBufferRx) )
   {

    gszBufferRx[gnTamBufferRx++] = c;
    #if(USA_TIMEOUT_SW == 1)
    if( !(gnTamBufferRx%10) )
    {
     //log("[reiniciando timer Rx]");
```

```
        xTimerStart ( gxTimerRecepcao, 0);
       }
       #endif
      }
      else
      {
       //acesso exclusivo via semáforo ao buffer
       logval("cheio - Qtd bytes: ", gnTamBufferRx);
       serial_copiarRxParaFila();
      }
    }
    xSemaphoreGive( SemaforoAcessoBufferRx );
  }
```

Además de este mecanismo de *timeout* de espera, la función evita que la sección crítica use el semáforo `SemaforoAcessoBufferRx` , que se crea sin semaphore_init ().

Si el búfer se llena, su contenido se copia en la cola y se siguen recibiendo nuevos caracteres en el búfer de reinicio. Esto se hace llamando a la función `serial_copiarRxParaFila()` .

Sin embargo, si el búfer no está lleno pero la recepción se interrumpe (ya sea debido a una falla de comunicación o al final del paquete), el tiempo de espera se agota y freeRTOS llama a la función `callbackTimeoutEntreCaracteresRX()` :

```
  void callbackTimeoutEntreCaracteresRx( TimerHandle_t xTimer)
  {
    //… detalhes da função omitidos aqui
    logval("timeout - Qtd bytes: ", gnTamBufferRx);
```

```
    serial_copiarRxParaFila();
    //… detalhes da função omitidos aqui
}
```

Se han omitido algunos detalles de la función para resaltar que el *timeout* de espera básicamente llama a la misma función de copia para la cola de recepción. Esta función a su vez hace el trabajo de colocar los elementos del *buffer* en la cola y reiniciarlo. El siguiente es el recorte de la función con números de línea para una mejor referencia, al igual que las llamadas freeRTOS están en negrita:

```
001: void serial_copiarRxParaFila(void)
002: {
003:   int i;
004:   int iniciou = 0;
005:   int tentativas = 5;
006:
007:   log("-------->");
008:   if(gnTamBufferRx == 0)
009:   {
010:     log("a");
011:     #if(USA_TIMEOUT_SW == 1)
012:     xTimerStart ( gxTimerRecepcao, 0);
013:     #endif
014:     return;
015:   }
016:
017:   log("b");
018:   for(i=0; i<gnTamBufferRx; i++)
019:   {
020:     log("c");
021:     if(!iniciou && !gszBufferRx[i])
022:     {
023:       log("d");
024:       continue;
025:     }
```

```
026:
027:     log("e");
028:     iniciou = 1;
029:     if( xQueueSendToBack(gxColaRecepcao, (void
*)&gszBufferRx[i], 1000 / portTICK_PERIOD_MS /*portMAX_DELAY*/)
== pdPASS )
030:     {
031:       log("*");
032:       log("f");
033:
034:       tentativas = 5;
035:
036:       gszBufferRx[i] = 0; //zera aquo ao inves do em um loop
037:
038:       if( i == (gnTamBufferRx-1) )
039:       {
040:         log("g");
041:
042:         // manda 0 para sinalizar fim de linha na tesk que
junta em linhas
043:         gszBufferRx[i] = '\0';
044:
045:         xQueueSendToBack(gxColaRecepcao, (void
*)&gszBufferRx[i], 0);
046:         log("h");
047:
048:         gnTamBufferRx = 0;
049:         break;
050:       }
051:     }
052:     else
053:     {
054:       log("i");
055:       //eperando infinitamente não perderia, se não esperar,
zera contador aqui, ficando o que botou na fila
056:
057:       //e perdendo o resto
```

```
058:       if(!tentativas--)
059:       {
060:         gnTamBufferRx = 0;
061:
062:          log("J");
063:
064:         break;
065:       }
066:     }
067:   }
068:
069:   log("--- k ---");
070:
071: }
```

Hay varias llamadas a la macro log () que pueden apuntar a un envío a serial0 para fines de registro, volviéndose gris como comentarios.

¿Cuándo debería terminar un libro?

Esa es una pregunta muy difícil. Esto se detiene aquí, aunque sin contemplar todos los freeRTOS (lo cual es comprensible). Pero tiene elementos muy útiles para que pueda realizar sus aplicaciones reales con características muy ricas, que tendrá a su disposición como navaja suiza.

Pero solo usando cada herramienta de este interruptor imaginario perfeccionará sus habilidades y le dará experiencia en las mejores aplicaciones.

Mi sugerencia es pasar a freeRTOS para nuevos proyectos, buscando conocimiento en el sitio web de freeRTOS (https://www.freertos.org/Documentation/RTOS_book.html):

Mastering the FreeRTOS Real Time Kernel - a Hands On Tutorial Guide

FreeRTOS V10.0.0 Reference Manual

Book companion source code

Asegúrese de revisar su libro y, si lo prefiere, estoy disponible en max.back@google.com con el tema del libro (para poder identificar mejor el correo electrónico).

Apéndice A - Fuentes adicionales

En este apéndice, tenemos listados completos que no se han abordado por completo (o son el resultado de mejoras en los listados listados) que se pueden copiar según el lector que se esté utilizando. Sin embargo, se enumeran aquí para un examen más fácil.

Apéndice A.1 - AnalogRead_DigitalREadModifiedVariousReaders (con suspensión de tasks)

```
   #include <Arduino_FreeRTOS.h>
    #include <semphr.h>  // add the FreeRTOS functions for Semaphores (or Flags).

   //tipo de parametros
   typedef struct {
     const char *pcNomeTask; //nome da task A SER CRIADA
     const char *pcTexto; / /texto a ser enviado pela serial antes da leitura, identificando sua fonte
     int SensorID; //ID da entrada analógica usada
     int TempoDelayTicks; / /tempo entre envio de leituras, em ticks
   } AnalogReadParametro_t;

   // Declare a mutex Semaphore Handle which we will use to manage the Serial Port.
   // It will be used to ensure only only one Task is accessing this resource at any time.
   SemaphoreHandle_t xSerialSemaphore;

   // define two Tasks for DigitalRead & AnalogRead
   void TaskDigitalRead( void *pvParameters );
   //aceita como parametro AnalogReadParametro_t
   void TaskAnalogReadParam( void *pvParameters );

   //define parametros para três tasks de leitura analógica que aceitam paramtros de estrutura
   AnalogReadParametro_t xParams[3], *pxParam = xParams;


   // the setup function runs once when you press reset or power the board
   void setup() {

     // initialize serial communication at 9600 bits per second:
    Serial.begin(9600);

    while (!Serial) {
      ; // wait for serial port to connect. Needed for native USB, on LEONARDO, MICRO, YUN, and other 32u4 based boards.
```

```
    }

     / / Semaphores are useful to stop a Task proceeding, where
it should be paused to wait,
    // because it is sharing a resource, such as the Serial
port.
    // Semaphores should only be used whilst the scheduler is
running, but we can set it up here.
     if ( xSerialSemaphore == NULL )  // Check to confirm that
the Serial Semaphore has not already been created.
    {
      xSerialSemaphore = xSemaphoreCreateMutex();  // Create a
mutex semaphore we will use to manage the Serial Port
     if ( ( xSerialSemaphore ) != NULL )
       xSemaphoreGive( ( xSerialSemaphore ) );  // Make the
Serial Port available for use, by "Giving" the Semaphore.
    }


      //define parametros para tres tasks de leitura analógica
que acietam parametros de estrutura

     pxParam->pcNomeTask = "TaskEntradaA1";
     pxParam->pcTexto = "Entrada A1";
     pxParam->SensorID = A1;
     pxParam->TempoDelayTicks = pdMS_TO_TICKS( 250UL );
     pxParam++;

     pxParam->pcNomeTask = "TaskEntradaA2";
     pxParam->pcTexto = "Entrada A2";
     pxParam->SensorID = A2;
     pxParam->TempoDelayTicks = 1; //pdMS_TO_TICKS( 250UL );
     pxParam++;

     pxParam->pcNomeTask = "TaskEntradaA3";
     pxParam->pcTexto = "Entrada A3";
     pxParam->SensorID = A3;
     pxParam->TempoDelayTicks = 1; //pdMS_TO_TICKS( 250UL );
     pxParam++;


      //xTaskCreate(TaskAnalogReadParam, pxParam->pcNomeTask,
128, (void *)&xParams[0], 1  /* Priority  */, NULL );
     //xTaskCreate(TaskAnalogReadParam, pxParam->pcNomeTask,
128, (void *)&xParams[1], 1  /* Priority  */, NULL );
```

```
      //xTaskCreate(TaskAnalogReadParam, pxParam->pcNomeTask,
128, (void *)&xParams[2], 1  /* Priority  */, NULL );

       //armazena a última task criada
      TaskHandle_t xUltimaTask;
       //cria as tasks percorre array de ponteiros até o nulo
(para facilitar)
      for(pxParam = &xParams[0]; pxParam < &xParams[3];
pxParam++)
      {
        Serial.println("\n--------------------------------");
        Serial.print("Criando ");
        Serial.println(pxParam->pcNomeTask);

        xTaskCreate(TaskAnalogReadParam, pxParam->pcNomeTask,
128, (void *)pxParam, (configMAX_PRIORITIES - 2)  /*
Priority  */ , &xUltimaTask );
      }
      vTaskSuspend(xUltimaTask);

       // Now set up two Tasks to run independently.
      xTaskCreate(
        TaskDigitalRead
        ,  (const portCHAR *)"DigitalRead"  // A name just for
humans
        ,  128  // This stack size can be checked & adjusted
by reading the Stack Highwater
        ,  (void *)xUltimaTask //Passa o valor do hadle da task
com cast para (void *)
        ,  (configMAX_PRIORITIES - 3)  // Priority, with 3
(configMAX_PRIORITIES - 1) being the highest, and 0 being the
lowest.
        ,  NULL );

   // Now the Task scheduler, which takes over control of
scheduling individual Tasks, is automatically started.
   }
   void loop()
   {
     // Empty. Things are done in Tasks.
   }
```

```
   void SerialDebugComSemaforo(const char *pszTexto)
   {
     if ( xSemaphoreTake( xSerialSemaphore, ( TickType_t ) 5 )
== pdTRUE )
     {
       Serial.print(pszTexto);

       xSemaphoreGive( xSerialSemaphore ); // Now free or
"Give" the Serial Port for others.
     }
   }

   /*-----------------------------------------------------*/
   /*---------------------- Tasks ----------------------*/
   /*-----------------------------------------------------*/

    void TaskDigitalRead( void *pvParameters)  // This is a
Task.
   {
     /*
     DigitalReadSerial
     Reads a digital input on pin 2, prints the result to the
serial monitor

     This example code is in the public domain.
    */

    // digital pin 2 has a pushbutton attached to it. Give it a
name:
    uint8_t pushButton = 2;

    TaskHandle_t xTaskParaReiniciar = (TaskHandle_t)
pvParameters;

     // make the pushbutton's pin an input:
    pinMode(pushButton, INPUT);

     for (;;) // A Task shall never return or exit.
    {
       // read the input pin:
     int buttonState = digitalRead(pushButton);

       // See if we can obtain or "Take" the Serial Semaphore.
     // If the semaphore is not available, wait 5 ticks of the
Scheduler to see if it becomes free.
     if ( xSemaphoreTake( xSerialSemaphore, ( TickType_t ) 5 )
== pdTRUE )
     {
```

```
        //Envia o texto recebido por parâmetro
        Serial.print("entrada digital: ");

        // We were able to obtain or "Take" the semaphore and
can now access the shared resource.
        // We want to have the Serial Port for us alone, as it
takes some time to print,
        // so we don't want it getting stolen during the middle
of a conversion.
        // print out the state of the button:
        Serial.println(buttonState);

        xSemaphoreGive( xSerialSemaphore ); // Now free or
"Give" the Serial Port for others.

        //Decide se reinicia a task
        if( (xTaskParaReiniciar != NULL) && (buttonState == 1)
)
        {
         vTaskResume(xTaskParaReiniciar);
        }

      }

       vTaskDelay(1);  // one tick delay (15ms) in between
reads for stability
    }
   }


    void TaskAnalogReadParam( void *pvParameters )  // This is
a Task.
   {
      //pegando o parametro como o texto da string
     AnalogReadParametro_t *pxParams = (AnalogReadParametro_t *)
pvParameters;

     for (;;)
     {
       // read the input on analog pin:
      int sensorValue = analogRead(pxParams->SensorID);

       // See if we can obtain or "Take" the Serial Semaphore.
      // If the semaphore is not available, wait 5 ticks of the
Scheduler to see if it becomes free.
```

```
      if ( xSemaphoreTake( xSerialSemaphore, ( TickType_t ) 5 )
== pdTRUE )
      {
        //Envia o texto recebido por parâmetro
        Serial.print(pxParams->pcTexto);
        Serial.print(": ");
        // We were able to obtain or "Take" the semaphore and
can now access the shared resource.
        // We want to have the Serial Port for us alone, as it
takes some time to print,
        // so we don't want it getting stolen during the middle
of a conversion.
        // print out the value you read:
        Serial.println(sensorValue);

        xSemaphoreGive( xSerialSemaphore ); // Now free or
"Give" the Serial Port for others.

      }

       vTaskDelay(pxParams->TempoDelayTicks);  // one tick
delay (15ms) in between reads for stability
    }
   }
```

Apéndice A.2 - Ejercicio propuesto en el Capítulo 5

```
#include <Arduino_FreeRTOS.h>
#include <queue.h>

const byte PinoChave = 2;
//Nova fila para receber envetos da chave
QueueHandle_t xColaEventosChave;
//Identificador da fila
QueueHandle_t xCola;

//Função da nova task para receber envetos da chave
void TaskEventosChave( void *pvParametros );
//Função da task para notificar o ligamento da chave
void TaskEnvio( void *pvParametros );

void setup() {
 BaseType_t xRet;

 Serial.begin(9600);
 attachInterrupt(digitalPinToInterrupt(PinoChave),
  tratarInterrupcaoPinoChave, RISING);
 pinMode(PinoChave, INPUT_PULLUP);

 // Cria a fila para eventos da chave COM ITEM DE CONTADOR DE
TICKS
 xColaEventosChave = xQueueCreate(
   10,                  // Tamanho da fila: 10
   sizeof( TickType_t )   // Tamanho do item: Tamanho do tipo
 );

 if(xColaEventosChave) Serial.println("Fila xColaEventosChave
criada com sucesso!");

 // Cria a fila de envio de caracteres
 xCola = xQueueCreate(
   10,                  // Tamanho da fila: 10
   21 *sizeof( char )   // Tamanho do item: 21
 );

 if(xCola) Serial.println("Fila criado com sucesso!");

 if(xTaskCreate(TaskEventosChave, (const portCHAR *)
"TaskEventosChave",  128, NULL, 2, NULL) == pdPASS)
 {
   Serial.println("Task TaskEventosChave criada com sucesso:");
 }

 if(xRet == pdPASS) Serial.println("Task criada com sucesso:");
```

```
 xRet = xTaskCreate(TaskEnvio, (const portCHAR *) "TaskEnvio",
 128, NULL, 2, NULL);

 if(xRet == pdPASS) Serial.println("Task criada com sucesso:");
}

void loop() {
 //Monitoramento da flag saiu daqui
}

void tratarInterrupcaoPinoChave() {
 BaseType_t xHigherPriorityTaskWoken;
 TickType_t xContadorTicks;

 xContadorTicks = xTaskGetTickCountFromISR();
 //coloca o valor de ticks atual na fila
 xQueueSendToBackFromISR( xCola, &xContadorTicks,
&xHigherPriorityTaskWoken);
 if( xHigherPriorityTaskWoken )
 {
  taskYIELD ();
 }
}

//Função da task para tratar evento da chave
void TaskEventosChave( void *pvParametros ) {
 TickType_t xtempoUltimaInterrupcao;
 TickType_t xAhora;
 int nTempoEmMs;

 //Armazenada o texto que será recebido
 char szTexto[21];

 Serial.println("Inciando loop: TaskEventosChave");

 //coloca o contador no inicio como se fosse da primeira int.
 xtempoUltimaInterrupcao = xTaskGetTickCount();

 //loop da task
 for( ;; )
 {
  //Espera indefinidamente até ler um item da fila
  xQueueReceive( xColaEventosChave, &xAhora, portMAX_DELAY );

  nTempoEmMs = (xAhora - xtempoUltimaInterrupcao) *
portTICK_PERIOD_MS;
```

```
   sprintf(szTexto, "Botão: %d ms", nTempoEmMs);
   xQueueSendToBackFromISR( xCola, szTexto, portMAX_DELAY);

   xtempoUltimaInterrupcao = xAhora;
  }
}

//Função da task para notificar o ligamento da chave
void TaskEnvio( void *pvParametros ) {
  //Armazenada o texto que será recebido
  char szTexto[21];

  Serial.println("Inciando loop: TaskEnvio");

  //loop da task
  for( ;; )
  {
    //Espera indefinidamente até ler um item da fila
   xQueueReceive( xCola, &szTexto, portMAX_DELAY );

   Serial.print(szTexto);
  }
}
```

Apéndice A.3 - Fuente compacta del Listado 9 para copiar al IDE

```
#include <Arduino_FreeRTOS.h>
#include <queue.h>

typedef enum {tiValorLecturaTemperatura, tiHoras,
tiTextoRecebido} TipoItem_t;
// Tipo de los tres tipos de datos (usando la unión) para
interpretar la memoria de una especie
typedef struct {
 TipoItem_t Tipo;
 union {
  float Temperatura;
  struct {
   uint8_t Hora;
   uint8_t Minutos;
   uint8_t Segundos;
  } Horario;
  char Texto[13];
 } Dados;
} ItemFila_t;

//Identificador da fila unica
QueueHandle_t xCola;

void TaskTomarLecturaTemp( void *pvParametros );
void TaskGenerarHoraActual( void *pvParametros );
void TaskRecepcaoSerial( void *pvParametros );
void TaskEnvio( void *pvParametros );

int criarTasks(void)
{
  if(xTaskCreate(TaskLeituraTemp, (const portCHAR *)
"TaskLeituraTemp",  128, NULL, 2, NULL) == pdPASS)
   Serial.println("Task TaskLeituraTemp criada com sucesso!");
  else
   return 0;

  if(xTaskCreate(TaskHoraAtual, (const portCHAR *)
"TaskHoraAtual",  128, NULL, 2, NULL) == pdPASS)
   Serial.println("Task TaskHoraAtual criada com sucesso!");
  else
   return 0;

  if(xTaskCreate(TaskRecepcaoSerial, (const portCHAR *)
"TaskRecepcaoSerial",  128, NULL, 2, NULL) == pdPASS)
```

```
    Serial.println("Task TaskRecepcaoSerial criada com
sucesso!");
   else
    return 0;

   if(xTaskCreate(TaskEnvio, (const portCHAR *)
"TaskEnvio",  128, NULL, 2, NULL) == pdPASS)
    Serial.println("Task TaskEnvio criada com sucesso!");
   else
    return 0;

   return 1;
}

void setup() {
 BaseType_t xRet;
 Serial.begin(9600);

 xCola = xQueueCreate(10, sizeof( ItemFila_t ));
 if(xCola)
 {
  Serial.println("Fila criado com sucesso!");
  if(!criarTasks())
   Serial.println("Ocorreu erro ao criar as tasks!");
 }
 else
  Serial.println("Ocorreu erro ao criar a fila!");
}

void loop() {
}

void TaskTomarLecturaTemp( void *pvParametros )
{
 ItemFila_t Item;
 Item.Tipo = tiValorLecturaTemperatura;

 Serial.println("Inciando loop: TaskLeituraTemp");
 for( ;; )
 {
  int sensorValue = analogRead(A0);
  Item.Datos.Temperatura = (float(sensorValue)*5/(1023))/0.01;
  xQueueSendToBack(xCola, &Item, portMAX_DELAY);
  vTaskDelay(500 / portTICK_PERIOD_MS);
 }
}

void tratarRelogio(ItemFila_t *pItem)
{
   pItem->Dados.Horario.Segundos++;
```

```
   if(pItem->Dados.Horario.Segundos == 60)
   {
    pItem->Dados.Horario.Segundos = 0;
    pItem->Dados.Horario.Minutos++;

    if(pItem->Dados.Horario.Minutos == 60)
    {
     pItem->Dados.Horario.Minutos = 0;
     pItem->Dados.Horario.Hora++;

     if(pItem->Dados.Horario.Hora == 24)
      pItem->Dados.Horario.Hora = 0;
    }
   }
}

void TaskGenerarHoraActual( void *pvParametros )
{
 ItemFila_t Item;
 Item.Tipo = tiHoras;
 Item.Datos.Horario.Hora = 0;
 Item.Datos.Horario.Minutos = 0;
 Item.Datos.Horario.Segundos = 0;

 Serial.println("Inciando loop: TaskRecepcaoSerial");
 for( ;; )
 {
  tratarRelogio(&Item);
  xQueueSendToBack(xCola, &Item, portMAX_DELAY);
  vTaskDelay(1000 / portTICK_PERIOD_MS);
 }
}

void TaskRecepcaoSerial( void *pvParametros )
{
 ItemFila_t Item;
 char carLido;
 int tamanho = 0;

 Item.Tipo = tiTextoRecebido;
 Item.Datos.Texto[tamanho] = '\0';

 Serial.println("Inciando loop: TaskRecepcaoSerial");
 for( ;; )
 {
  if(Serial.available())
  {
   carLido = (char)Serial.read();
   if( carLido != '\n')
   {
```

```
     Item.Datos.Texto[tamanho++] = carLido;
     Item.Datos.Texto[tamanho] = '\0' ;
    }
    if( (carLido == '\n') || (tamanho ==
(sizeof(Item.Datos.Texto)-1)) )
    {
      xQueueSendToBack(xCola, &Item, portMAX_DELAY);
      tamanho = 0;
      Item.Datos.Texto[tamanho] = '\0';
    }
  }
 }
}

void TaskEnvio( void *pvParametros ) {
 char EspejoDeDisplay[2][17] = {"00:00:00 |      ",
                       "T: 000.0 |      "};
 ItemFila_t Item;
 int l,i;
 char *p;

 Serial.println("Inciando loop: TaskEnvio");

 for( ;; )
 {
  xQueueReceive( xCola, &Item, portMAX_DELAY );
  switch(Item.Tipo)
  {
   case tiValorLecturaTemperatura:
    sprintf(&EspejoDeDisplay[1][3], "%3.1f",
Item.Datos.Temperatura);
    break;
    case tiHoras:
     sprintf(&EspejoDeDisplay[0][0], "%0.2d",
      Item.Datos.Horario.Hora);
     sprintf(&EspejoDeDisplay[0][3], "%0.2d",
      Item.Datos.Horario.Minutos);
     sprintf(&EspejoDeDisplay[0][6], "%0.2d",
      Item.Datos.Horario.Segundos);
    break;
    case tiTextoRecebido  :
     for(l=0; l<2; l++)
     {
      for(i=0, p = &EspejoDeDisplay[l][10]; i<6; i++, p++) *p =
' ';
       *p = '\0';
     }

     for(l=0; l<2; l++)
```

```
     {
      p = &EspejoDeDisplay[l][10];
      for(i=0; i<6 && Item.Datos.Texto[i]; i++)
      {
         *p = Item.Datos.Texto[i];
         p++;
      }
      *p = '\0';
     }
    break;
   }
   Serial.println("----------------");
   Serial.println(EspejoDeDisplay[0]);
   Serial.println(EspejoDeDisplay[1]);
  }
}
```

Apéndice A.4 - Ejemplos de métodos de llamada CanalEntreTasks

En este apéndice tenemos fragmentos de código "arrojado" de los usos de la clase CanalEntreTasks y en github puede ver estos códigos de grupo, utilizados para comunicarse entre tareas con estructuras de soporte llamadas tuberías, es decir, pares de colas de entrada y salida en nodos y tareas que utilizan estas estructuras para leer desde un canal, procesar los datos y colocarlos en otro canal.

Este código está en el enlace. https://github.com/maxback/multitarefa_na_pratica/tree/master/CanaisSemelhanteAoGoLang , si quieres caminar o incluso descargarlo y ver su funcionamiento en acción.

Pero siga los fragmentos de código:

```
#include "CanalEntreTasks.h"

//tipo de parametros
typedef struct {
 const char *pcNomeTask;
 const char *pcTexto;
 int SensorID;
 int TempoDelayTicks;
 QueueHandle_t xColaCalculo;
 CanalEntreTasks<int> *CanalEnvio;
} AnalogReadParametro_t;

CanalEntreTasks<int> *CanalEnvio;

...
     if(pxParams->CanalEnvio != NULL)
     {
      //pxParams->CanalEnvio->enviar(sensorValue);

      //Teste com o operador de override:
      /*
      //envia para o canal
      *(pxParams->CanalEnvio) << 123;
      //e sobrescreve em seguida
      *(pxParams->CanalEnvio) < sensorValue;
```

```
          */


          *(pxParams->CanalEnvio) << sensorValue;

          //poderiam ser varios valores
          //*(pxParams->CanalEnvio) << sensorValue-1 <<
sensorValue << sensorValue+1;

          /*
          (*pxParams->CanalEnvio) << 123;

          operator<<(   operator<<((*pxParams->CanalEnvio), 99),
sensorValue) ;

          operator<<((*pxParams->CanalEnvio),
99).enviar(sensorValue);
          operator<<((*pxParams->CanalEnvio),
sensorValue).enviar(123);


          (*pxParams->CanalEnvio) << sensorValue << 123;
          */
        }
      }


   typedef struct {
    CanalEntreTasks<int> *entrada;
    CanalEntreTasks<int> *saida;
   } CanaisDaTask_t;

   // Define 3 canais e 3 funcoes de tasks
   CanalEntreTasks<int> xCanalMultiplica(10,
"CanalEntMultiplica");
   //CanalEntreTasks<int> xCanalDiminui(10);
   //CanalEntreTasks<int> xCanalImprime(30);

   CanaisDaTask_t C1 = {&xCanalMultiplica, new
CanalEntreTasks<int>(10, "CanalEntDiminui")};
   CanaisDaTask_t C2 = {C1.saida, new CanalEntreTasks<int>(30,
"CanalImprime")};
   CanaisDaTask_t C3 = {C2.saida, NULL};

      //valor = C->entrada->receber();
      *(C->entrada) >> valor;

      SerialDebugComSemaforo("\nTaskMultiplica recebeu ",
valor);
       *(C->saida) << valor;
      *(C->entrada) >> valor;
```

```
        //C->saida->enviar(diminuido);
       *(C->saida) << valor;

        SerialDebugComSemaforo("\n   -> DEPOIS TaskDiminui
enviando para", C->saida->getNome(), ": ",  valor);

     //seta timeout desta fila
     C->entrada->setTempoTimeoutMs(100);

      *(C->entrada) >> valor;

      //*(C->entrada) >> valor;

        *(C->saida) << valor;


        *(pipeline_getCanalEntrada()) << valor;
```

Apéndice B - Notas del seminario web freeRTOS (Microgenios y embacados.com.br)

En este apéndice tenemos información obtenida de la audiencia de un seminario web promovido por Microgenios y embarados.com.br con el profesor Fernando Simplício. En este seminario web, cuya memoria está hecha de la captura de pantalla de diapositivas (realizada para registrar información pasada), enfatizó aspectos positivos del uso de estos sistemas y alentó fuertemente su uso y popularización en Brasil, abordando principalmente los siguientes puntos, recreados desde lo más fiel posible dada su riqueza e importancia:

Tabla - Temas principales del seminario web.

Titulo	Resumen
Proyecto con MCU	Los diseños de MCU a menudo se caracterizan por la conexión de varios sensores, actuadores, módulos de comunicación y dispositivos de salida como LCD.
Programa C	Presentaba la forma en que generalmente se hacen los programas C para MCU. Primero abordó el caso en el que el bucle principal llama a una función para realizar la tarea de manejar cada dispositivo (o los datos guardados al respecto en el tratamiento de su interrupción), ya sea que necesite ser tratado o no, es decir, el método de sondeo de los dispositivos. dispositivos. Este tipo de procedimiento lleva mucho tiempo reparar un sensor en particular, ya que tiene que atender a todos los demás antes de volver a la función de este sensor.
Máquina de estado	Una alternativa para minimizar esto (sin usar un programador) es dividir las funciones de tareas largas en varias funciones más pequeñas que dividen la tarea en pasos de acuerdo con una máquina de estados. Esto acorta un poco el tiempo de manejo ya que cada iteración del ciclo principal

solo se ejecuta un estado (o paso de su procesamiento) de la tarea, luego la siguiente tarea.

Compartir recursos

Aborda el hecho de que los diversos archivos .h y .c creados para modularizar las porciones de software incorporado para capacidades de hardware. Estos recursos a menudo se comparten y usan al incluir directamente la aplicación principal (main.h y main.c).

Conductores de recursos

El orador comienza a mostrar cómo varias partes del sistema pueden compartir recursos en un RTOS dividiendo el trabajo en tareas (tareas) del sistema operativo. Los controladores (llamados controladores armónicos) se escriben teniendo en cuenta la competencia por sus características y utilizan mecanismos del sistema operativo como los semáforos para armonizar (en el ejemplo dado) dos llamadas a una función de controlador que envía un texto a través de la serie. Si el escenario sugerido es el siguiente:

- Se produce algún evento y la Tarea A comienza a ejecutarse, de menor prioridad, llama a la función del controlador USART para escribir el texto "Hola Palabra". Esto inicia la transmisión y antes de terminar de enviar la tarea B comienza a ejecutarse (debido a otro evento, junto con el hecho de que la Tarea B tiene mayor prioridad: prioridad).
- La tarea B también llama a la función del controlador para escribir el texto "Go Rattlers!", Pero como la función está bloqueada por el controlador, el sistema operativo hace que la tarea se bloquee sin iniciarla. enviando la segunda cadena. Una vez que la Tarea B ha sido bloqueada, la Tarea A se vuelve a ejecutar;
- La tarea A reanuda la ejecución y espera a que se complete la escritura antes de que finalice. Tan pronto como la función del controlador libera el recurso, al liberar el semáforo (por ejemplo) RTOS hace que la tarea B asuma la ejecución mientras espera el recurso y tiene mayor prioridad.

• La tarea B reanuda la ejecución y la función de escritura finalmente ejecuta la operación.

Esta diapositiva ejemplifica cómo RTOS se preocupa por ciertos detalles, mientras que cada función se desempeña como si fuera la única que escribe cosas en serie.

En multitarea debemos considerar (...)

- • El tiempo para el cambio de contexto.
- • Consumo de memoria para almacenamiento de contexto.
- • Las tareas pueden ser tolerantes con bucles infinitos.
- • Compartir recursos (registros internos, memoria, unidades lógicas aritméticas de CPU y periféricos como LCD, sensores y actuadores), entre otras tareas que compiten.

Ejemplo de programa multitarea Proporciona un breve ejemplo de cómo un programa multitarea funcionaría en un sistema no preventivo.

Multitaskin g policy Ejemplificó cómo el *scheduler* determina las reglas de conmutación (como comentamos en este documento).

multitasking preemptive Muestra un ejemplo en el que las ocurrencias de interrupción provocan la preferencia entre dos tareas, mostrando la acción del planificador y el cambio de contexto al guardar y restaurar los registros de control de ejecución del microcontrolador.

Secciones Críticas Comentó sobre el tema de las secciones críticas (ya hemos comentado sobre esto en este documento)

Recreo (“*re-enter* ”)

- • Denota problemas que pueden ocurrir cuando el mismo código se ejecuta simultáneamente por

múltiples tareas o cuando se accede a datos globales simultáneamente por múltiples tareas.

En un entorno *multitasking* , las funciones deben ser preferiblemente reentrantes.

Seções críticas

Luego, el orador muestra un ejemplo de una sección crítica, que ejemplifica el código PIC para realizar una operación de byte O entre el valor del puerto PORTB y el valor 01 en hexadecimal. Primero se muestra en lenguaje C, ocupando una línea y su equivalente de ensamblaje, que ocupa cuatro líneas y describe las instrucciones realmente ejecutadas.

Esto es para ejemplificar el acceso no atómico, que requiere varios comandos de máquina para ejecutarse. Como este puerto es un recurso compartido, y su acceso no es atómico, tenemos una función no reentrante y una sección crítica, es decir, una sección que si el programador cambia el contexto causará inconsistencia de operación en la otra tarea, o al regresar a El actual.

En otra diapositiva muestra una función que es reentrante, con las siguientes características:

• Recibe datos para trabajar a través de parámetros (que se pasan a través de la pila o a través de un registro especial de microcontrolador). Como cada tarea tiene su propia pila de destino y un conjunto de registros, el acceso al parámetro pasado no constituye acceso a algo compartido.

• También utiliza una variable local (dentro del alcance de la función) asignada para cada ejecución de la función, también en el área de pila o registros de la tarea que la llamó;

Conclusión: incluso si dos tareas lo llaman simultáneamente, cada tarea tendrá una copia del parámetro y una copia de la variable local, sin confusión entre los datos de las dos llamadas.

Secciones críticas

El ejemplo anterior es seguido por el ejemplo de una función no reentrante, porque acceder a una variable global del módulo, que es estática y no está asignada en la pila, es decir, todas las llamadas simultáneas tendrán acceso a un área de

	memoria única. funcionará en tareas separadas y se producirá corrupción de la información.
Ejemplo: Receso (“*re-enter* ”)	El orador ejemplifica el caso de error de concurrencia para una variable global a través de C y código de ensamblaje y un diagrama que muestra la programación entre dos tareas que acceden a la misma variable. Finalmente concluye: • Nunca debería haber acceso a datos globales simultáneos si al menos una de las tareas puede modificar estos datos. • Por lo tanto, se debe evitar el uso de datos globales compartidos. Si esto no es posible, se debe utilizar un semáforo para proteger estos datos del acceso simultáneo.
Beneficios de un programa. *Multitasking*	• Un procesador convencional solo puede realizar una tarea a la vez. Pero un sistema operativo multitarea puede hacer que parezca que cada tarea se ejecuta simultáneamente.
Schedure ; Colas y *Schedure* ; Colas y *Schedure* (w / prioridades); *Schedure* (w / prioridades); *Schedure* (tipos básicos);	Proporcionó una explicación del planificador utilizando colas y colas por prioridad, así como los tipos básicos de planificador (este segundo tema lo encontramos bien para agregar en la revisión de la literatura de este documento).
RTOS	Conceptualizó RTOS destacando su carácter fundamental: la cuestión del tiempo, es decir, las limitaciones de tiempo para

asistir a eventos que tiene que respetar, como ya hemos discutido.

Agregó el hecho interesante de que el software en tiempo real no tiene que ser multitarea.

Comercial RTOS

- Compatible con muchas plataformas de rendimiento pequeño y mediano.
- Los RTOS comerciales son ampliamente probados por sus desarrolladores y comunidad.

FreeRTOS

- Sistema operativo en tiempo real.
- Gratis y de código abierto.
- Desarrollado por Real Time Engineers Ltda.
- Cuenta con gestión de tareas y servicios de memoria.
- Diseñado para ser aplicado a MCU de pequeña capacidad de memoria (ARM7 ~ 4.3kbytes).
- Desarrollado en lenguaje C (puede compilarse en GCC, Borland C ++, etc.).
- Admite un número limitado de tareas y prioridades (según el hardware utilizado).
- Desafortunadamente, no viene con controladores, interfaces de comunicación de red, controladores o archivos (FileSystem) - (para algunas familias de MCU es gratis).
- Implementa colas, semáforos binarios, contadores, mutexes y servicios de temporizadores de software.
- Actualmente es compatible con más de 38 arquitecturas diferentes.

Después de enumerar estas características, explicó cómo funciona la estructura estandarizada de carpetas de distribución FreeRTOS y el esquema de *port* (refiriéndose a la portabilidad para una plataforma específica). Finalmente, lo guía para

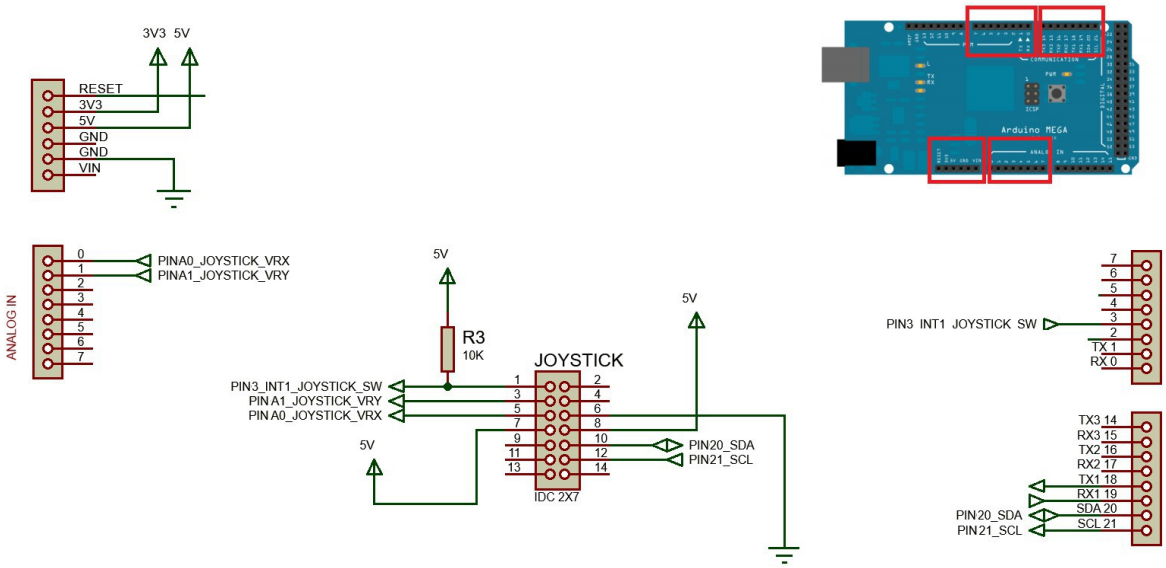
	comenzar a usar FreeRTOS a partir del ejemplo que viene con el puerto de plataforma deseado.
FreeRTOS	Continuó presentando el patrón de nombres de variables, parámetros y funciones, y explicó las diversas opciones de asignación de memoria dinámica (proporcionadas por RTOS en lugar de la función malloc () de C). Este sistema se utiliza tanto para la creación de tareas, colas y similares, así como para asignar la dinámica necesaria en el código del sistema que se está desarrollando. Este tema es demasiado profundo para agotarse y se puede encontrar en la documentación del proveedor a continuación.
Tasks	Finalmente, presentó y explicó un ejemplo de un código de creación de tareas en C y explicó sus diversos estados de ejecución.

Fonte: do autor

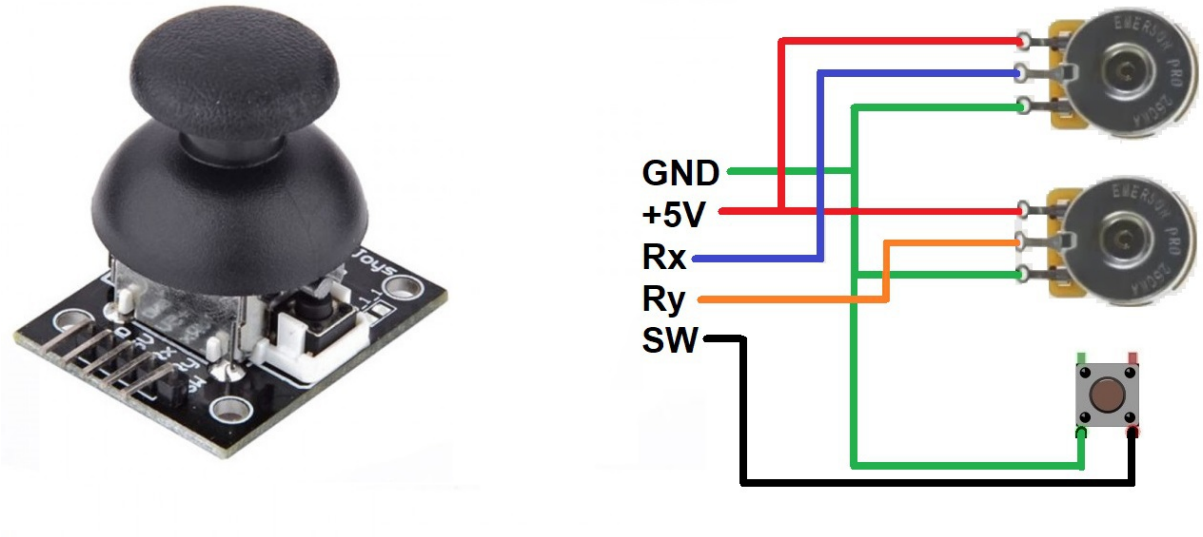
Apéndice C - Joystick y conector de expansión de canal I2C

En el esquema aparece un conector llamado joystick que desempeña el papel de conectarse a un joystick formado por dos potenciómetros, uno para el eje X y otros para el eje Y y un botón (que reacciona presionando el eje Z).

También incluye las líneas de comunicación de un canal I2C que se pueden usar para comunicarse con otros dispositivos, como una pantalla LCD, con el fin de mostrar los datos de estado en el propio dispositivo.



El montaje se puede hacer con un potenciómetro o con los potenciómetros asociados si no desea comprar este periférico:



El primer potenciómetro forma un divisor de voltaje entre 5V y 0V cuyo voltaje se entrega en Rx (eje X), que para nuestro circuito se conectará

al pin A0, es decir, a la entrada analógica 1 que convertirá este valor en valores entre 0 y 1023. El segundo potenciómetro hace lo mismo para el eje Y, conectando el resultado a la entrada analógica 1 (A1), con el mismo rango de valores.

Finalmente, el botón pulsador (*push-button*) conectado al eje Z (vertical) conectará Sw al GND cuando se presiona, y agregamos una resistencia conectando esta señal a + 5V, asegurando el nivel lógico 1 cuando no se presiona.

Apéndice D - Código fuente del proyecto práctico (segunda parte)

Si desea esta fuente comprimida, envíeme un correo electrónico a max.back@gmail.com con el tema "libro - fuentes".

ArduinoMega2560.ino

```
          #ifdef ARDUINO

#include <Arduino_FreeRTOS.h>
#include <semphr.h>
#include "FreeRTOSConfig.h"
#include <Arduino.h>
#include <HardwareSerial.h>
#include <Wire.h>

#else
#include <FreeRTOS.h>
#endif

#include "global.h"

#if(USAR_LCD == 1)
#include <LiquidCrystal_I2C.h>
#endif


#include "giroscopio.h"
#include "serial.h"
```

```
#include "botao.h"
#include "joystick.h"
#include "leds.h"
#include "util_memory.h"
#include "lcd.h"

#include "giroscopio.c"
#include "serial.c"
#include "botao.c"
#include "joystick.c"
#include "leds.c"
#include "util_memory.c"
#include "lcd.c"

#include "task_giroscopio.h"
#include "task_serial.h"
#include "task_saudacao.h"
#include "task_ecoserial.h"
#include "task_botao_usuario.h"
#include "task_joystick.h"
#include "task_cliente_http.h"
#include "util_memory.h"
```

```
#include "task_giroscopio.c"
#include "task_serial.c"
#include "task_saudacao.c"
#include "task_ecoserial.c"
#include "task_botao_usuario.c"
#include "task_joystick.c"
#include "task_cliente_http.c"
//#include "util_memory.c"

#include "aplicacao.c"

#define MOSTRAR_CAR_RECEBIDO 1
#define MOSTRAR_CAR_ENVIADO 0


#define MAX_QTD_ITENS_CALLBACK_LOOP 1
static int gnContadorItensLista = 0;


static callbackGenerico_t gpCalbackSerialEvent = NULL;

#if(USAR_LCD == 1)
LiquidCrystal_I2C lcd(0x27,2,1,0,4,5,6,7,3, POSITIVE);
#endif
```

```
/*-----------------------------------------------------------*/

/*

* Função chamada para inicializar o sistema com suas configurações e

* inicializações do módulos de alto nível e talbém das taks e demais

* coisas do freeRTOS.

*/

extern void vIniciarSistema( void );

/*-----------------------------------------------------------*/



/*

* Para inscrever funções que serão chaamdos no loop()

*/


/*

* Função de log na serial padrão

*/

void printlog(int qtdParams, const char *s, int valor)

{



  if(qtdParams == 1)
```

```
  {
    if(s[1] == '\0')
      Serial.write(s[0]);
    else
      Serial.println(s);
  }
  else
  if(qtdParams == 2)
  {
    Serial.print(s);
    if(s[1] == '\0')
      Serial.print(valor);
    else
      Serial.println(valor);
  }
}


/*
* Função de log na serial padrão
*/
void printlogss(const char *s, char * valor)
{
```

```
   Serial.print(s);
   if(s[1] == '\0')
     Serial.print(valor);
   else
     Serial.println(valor);


}


/*
* Funções para dar acesso a serial 1
*/
void ExecutarSerial1_begin(unsigned long baud)
{
  printlog(2, "Iniciando Serial1 com baud :", baud);
  Serial1.begin(baud);
}

int ExecutarSerial1_available(void)
{
   int result = Serial1.available();


   //printlog(2, "avalilable():", result);
```

```
  return result;
}

int ExecutarSerial1_read(void)
{
  int clido = Serial1.read();

#if(MOSTRAR_CAR_RECEBIDO == 1)
  Serial.write(clido);
#endif


  return clido;
}

void ExecutarSerial1_print(char *s)
{

#if(MOSTRAR_CAR_ENVIADO == 1)
  Serial.write('>');
  Serial.print(s);
#endif

   Serial1.print(s);
}
```

```
int ExecutarSerial1_testarEstahTransmitindo(void)

{

   return Serial1.availableForWrite() != (SERIAL_TX_BUFFER_SIZE
- 1);

}



void
ExecutarSerial1_registrarCalbackSerialEvent(callbackGenerico_t
pCallback)

{

  gpCalbackSerialEvent = pCallback;

}



void serialEvent1() {

  if(gpCalbackSerialEvent)

  {

    //Serial.write('.');

    gpCalbackSerialEvent(NULL);

  }

}


void wire_begin(void)

{
```

```
 Wire.begin();
}

void wire_writeRegister(int deviceAddress, byte address, byte
val)
{
   Wire.beginTransmission(deviceAddress); // start transmission
to device
   Wire.write(address);       // send register address
   Wire.write(val);         // send value to write
   Wire.endTransmission();     // end transmission
}

int wire_readRegister(int deviceAddress, byte address)
{
   int v;
   Wire.beginTransmission(deviceAddress);
   Wire.write(address); // register to read
   Wire.endTransmission();

   Wire.requestFrom(deviceAddress, 1); // read a byte

   while(!Wire.available())
   {
      // waiting
```

```
   }
   v = Wire.read();
   return v;
}

void lcd_begin(int colunas, int linhas)
{
#if(USAR_LCD == 1)
  Serial.print("A-lcd_begin(");
  Serial.print(colunas);
  Serial.write(',');
  Serial.print(linhas);
  Serial.println(") - inicio");


 lcd.begin(colunas,linhas);
 delay ( 1000 );
 Serial.println("A-lcd_begin() - fim");
#endif
}

void lcd_print(char *texto)
{
#if(USAR_LCD == 1)
```

```
  Serial.println("B-lcd_print() - inicio");
  lcd.print(texto);
  Serial.println("B-lcd_print() - fim");
#endif
}

void lcd_setCursor(int coluna, int linha)
{
#if(USAR_LCD == 1)
   Serial.print("C-lcd_setCursor(");
   Serial.print(coluna);
   Serial.write(',');
   Serial.print(linha);
   Serial.println(") - inicio");


   lcd.setCursor ( coluna, linha);


   Serial.println("C-lcd_setCursor() - fim");
#endif
}
```

```
void setup() {
  Serial.begin(9600);

  ////////////////////////////////
  lcd.begin(16,2);               // initialize the lcd
  lcd.home ();                   // go home
  lcd.print("Hello, ARDUINO ");
  lcd.setCursor ( 0, 1 );        // go to the next line
  lcd.print (" FORUM - fm   ");
  delay ( 1000 );
  ////////////////////////////////


  //Chama função para inicialzair o sistema
  vIniciarSistema();

}

void loop() {
  static int i = 0;
  static int j = 0;
  i++;

//Habilitar idle hook
```

```
//  if(!(i % 1000000))

//    Serial.println("idle...");



}



//Está faltando espeço apra alicar e não estou descobrindo como mudar no ambiente do Arduino o tamanho

//de memória heap para que o alocador padrão (heap_3.c funcione (que usa o malloc() por baixo dos panos).

//

//Vou mudar para o modelo heap_1 e definir o tamanho em configTOTAL_HEAP_SIZE


//para isso mudei o arquivo C:\Users\Max\Documents\Arduino\libraries\FreeRTOS\src\heap_3.c

//para ver ler o define ALOCACAO_JA_DEFINIDA_NO_SKETCK está indefinido, o que no meu caso desativa as implementações de lá e em outros deixa intacta

//coloquei os prototipo com o atributo __attribute__((weak)) em heap_3.c


//hadles de malloc e stack overflow como está o padrão em

//C:\Users\Max\Documents\Arduino\libraries\FreeRTOS\src\variantHooks.cpp

//piscando so leds mas também enviando mensagem pela serial.
```

```
void vApplicationStackOverflowHook( TaskHandle_t xTask
__attribute__((unused)), portCHAR *pcTaskName
__attribute__((unused))  )

{

 Serial.write('$');

 Serial.write(*pcTaskName);



 DDRB  |= _BV(DDB7);

 PORTB |= _BV(PORTB7);       // Main (red PB7) LED on. Main LED
on.


 leds_desliga(LEDS_LED_TODOS);


 if(*pcTaskName)

 {

  switch(*pcTaskName)

  {

   case TASKCLIENTEHTTP_NOME[0]:     //
TASKCLIENTEHTTP_NOME  "1ClienteHttp"

     leds_liga(LEDS_LED_VERMELHO);   //  \/

     leds_liga(LEDS_LED_AZUL);

     leds_liga(LEDS_LED_VERDE);

    break;
```

```
    case
TASKECOSERIAL_NOME[0]:    //  TASKECOSERIAL_NOME    "7AppEcoSer
ial"

      leds_liga(LEDS_LED_LARANJA);  //   '

    break;



    case
TASKSAUDACAO_NOME[0]:    //  TASKSAUDACAO_NOME     "6AppSaudaca
o"

      leds_liga(LEDS_LED_VERDE);  //  o

    break;


    case TASKJOYSTICK_NOME[0]:    //
TASKJOYSTICK_NOME     "5AppJoystick"

      leds_liga(LEDS_LED_AZUL);   //      .

    break;



    case TASKGIROSCOPIO_NOME[0]:   //
TASKGIROSCOPIO_NOME   "2AppGiroscopio"

      leds_liga(LEDS_LED_LARANJA);  //     /

      leds_liga(LEDS_LED_VERDE);

    break;



    case TASKSERIAL_TASTX_NOME[0]:  //  TASKSERIAL_TASTX_NOME
"3SerialTx"
```

```
      leds_liga(LEDS_LED_VERMELHO); //         o
    break;


    case TASKSERIAL_TASRX_NOME[0]:  //  TASKSERIAL_TASRX_NOME
"4SerialRx"
      leds_liga(LEDS_LED_LARANJA);   //
      leds_liga(LEDS_LED_VERMELHO);
    break;


    case TASKBOTAOSUAURIO_NOME[0]:  // TASKBOTAOSUAURIO_NOME
"0BotoUsu"
      leds_liga(LEDS_LED_AZUL);  //    >
      leds_liga(LEDS_LED_VERMELHO);
      leds_liga(LEDS_LED_LARANJA);
    break;

    case 'T':
      leds_liga(LEDS_LED_VERDE);  //    /\   //Timer?
      leds_liga(LEDS_LED_LARANJA);
      leds_liga(LEDS_LED_VERMELHO);
    break;

    default:
```

```
      leds_liga(LEDS_LED_LARANJA); //    <   desconhecido
      leds_liga(LEDS_LED_VERDE);
      leds_liga(LEDS_LED_AZUL);
   }
  }
  for(;;)
  {
   _delay_ms(2000);

   PINB  |= _BV(PINB7);       // Main (red PB7) LED toggle.
Main LED slow blink.
  }
}


void vApplicationMallocFailedHook( void )
{
  DDRB  |= _BV(DDB7);
  PORTB |= _BV(PORTB7);       // Main (red PB7) LED on. Main LED
on.
  Serial.write('%');
  Serial.write(')');
  Serial.println("Erro detectado ->
vApplicationMallocFailedHook");
  for(;;)
```

```
  {
    _delay_ms(50);


    PINB  |= _BV(PINB7);       // Main (red PB7) LED toggle.
Main LED fast blink.
  }
}
```

botao.c

```
          /*
* botao.c
*
*  Created on: 17 de MAIO de 2018
*      Author: Max
*/
//teste do caminho apontador via link simbÃ³lico e alterado na pasta alvo
// ** Baseado em: https://portal.vidadesilicio.com.br/usando-a-interrupcao-externa-no-seu-arduino/



#define _BOTAO_C_ARDUINO_ATIVO 1


#ifdef ARDUINO


#define _BOTAO_C_


#include "botao.h"
#include "tipos.h"


#include <Arduino_FreeRTOS.h>
#include <task.h>
```

```
#include <arduino.h>

#if (_BOTAO_C_ARDUINO_ATIVO == 1)
static QueueHandle_t gxColaNotificacao = NULL;
static TaskHandle_t gxTaskNotificacion = NULL;

static void EXTI0_IRQHandler(void);
static void EXTI1_IRQHandler(void);
#endif
/*
* @brief Inicializa o modulo
*/
void botao_init(void)
{
#if (_BOTAO_C_ARDUINO_ATIVO == 1)

gxColaNotificacao = NULL;
gxTaskNotificacion = NULL;

    //Configurando a interrupcao 0 e 1
   attachInterrupt(0, EXTI0_IRQHandler, RISING);
   attachInterrupt(1, EXTI1_IRQHandler, RISING);
#endif
}
```

```
/*
* @brief Configura ligacao do pressiona do botão
* com uma fila
* @param QueueHandle_t xColaNotificacao - Fila a ser notificada
* ele deve esperar um valor do tipo TickType_t que receberá o contador
* de ticks atual (para dar uma ideia de intervalo entre acionamentos)
*/
void botao_setFila(QueueHandle_t xColaNotificacao)
{
#if (_BOTAO_C_ARDUINO_ATIVO == 1)

gxColaNotificacao = xColaNotificacao;
#endif
}

/*
* @brief Configura task a ser nitificada do pressiona do botão
* com uma fila
* @param TaskHandle_t xTask - Task a ser notificada
* ele deve esperar um valor do tipo datos_lectura_boton_t nos 16 bits menos significativos
* a estrutura dos ID 0 (ou outros IDs pares, adicionais). Já nos 16 mais significativos
```

```
* do valor de notifcação virãos o ID 1 (ou outros IDs ímpares, adicionais)

* Uma notificação em cada grupo de bits sobrescreve a anterio, se nã ofo itratada ainda.

*/

void botao_setTaskParaNotificacoes(TaskHandle_t xTask)

{

#if (_BOTAO_C_ARDUINO_ATIVO == 1)

gxTaskNotificacion = xTask;

#endif

}



#if (_BOTAO_C_ARDUINO_ATIVO == 1)


/**

  * @brief  TTrata o interpção externa ligada ao botão

  * @param  None

  * @retval None

  */

void EXTI0_IRQHandler(void)

{


static TickType_t xUltimoTiempoInterrupcion = 0;
```

```
TickType_t xAhora;

BaseType_t xHigherPriorityTaskWoken = pdFALSE, xNotificouTask;

uint32_t ulValorTaskNotificion;

datos_lectura_boton_t leitura;

xAhora = xTaskGetTickCountFromISR();


leitura.nID = 0;

leitura.nTiempoEntreOcurrencias = xAhora -
xUltimoTiempoInterrupcion;

leitura.nFlagEventoOcurrido = 1;


xUltimoTiempoInterrupcion = xAhora;


if(gxTaskNotificacion)

{

ulValorTaskNotificion = leitura.nValor;

ulValorTaskNotificion <<= 0;

ulValorTaskNotificion &= 0x0000FFFF;


//Envia uma notificação diretamente para a task indicada por
gxTaskNotificacion

xNotificouTask = xTaskNotifyFromISR( gxTaskNotificacion,

       ulValorTaskNotificion, /* ulValue */

       eSetBits, /* eAction parameter. */

            &xHigherPriorityTaskWoken );
```

```
configASSERT( xNotificouTask == pdPASS );


}

else

if(gxColaNotificacao)

{

//se não couber na fila, perde (sem tratamento do retorno)

xQueueSendToBackFromISR(gxColaNotificacao, &leitura,
&xHigherPriorityTaskWoken);

}



   //portYIELD_FROM_ISR(xHigherPriorityTaskWoken);

   if( xHigherPriorityTaskWoken )

   {

   taskYIELD ();

   }

}


/**

  * @brief  TTrata o interpção externa ligada ao botão do
joystick

  * @param  None

  * @retval None

  */
```

```
void EXTI1_IRQHandler(void)

{

static TickType_t xUltimoTiempoInterrupcion = 0;

TickType_t xAhora;

BaseType_t xHigherPriorityTaskWoken = pdFALSE, xNotificouTask;

uint32_t ulValorTaskNotificion;

datos_lectura_boton_t leitura;



xAhora = xTaskGetTickCountFromISR();


leitura.nID = 1;

leitura.nTiempoEntreOcurrencias = xAhora -
xUltimoTiempoInterrupcion;

leitura.nFlagEventoOcurrido = 1;


xUltimoTiempoInterrupcion = xAhora;


if(gxTaskNotificacion)

{

ulValorTaskNotificion = leitura.nValor;

ulValorTaskNotificion <<= 16;

ulValorTaskNotificion &= 0xFFFF0000;
```

```
//Envia uma notificação diretamente para a task indicada por
gxTaskNotificacion

xNotificouTask = xTaskNotifyFromISR( gxTaskNotificacion,

       ulValorTaskNotificion, /* ulValue */

       eSetBits, /* eAction parameter. */

            &xHigherPriorityTaskWoken );


configASSERT( xNotificouTask == pdPASS );


}

else

if(gxColaNotificacao)

{

//se não couber na fila, perde (sem tratamento do retorno)

xQueueSendToBackFromISR(gxColaNotificacao, &leitura,
&xHigherPriorityTaskWoken);

}


   //portYIELD_FROM_ISR(xHigherPriorityTaskWoken);

   if( xHigherPriorityTaskWoken )

   {

   taskYIELD ();

   }

}

#endif
```

```
#endif
```

FreeRTOSConfig.h

```
          /*
* FreeRTOS Kernel V10.0.0
* Copyright (C) 2017 Amazon.com, Inc. or its affiliates.  All Rights Reserved.
*
* Permission is hereby granted, free of charge, to any person obtaining a copy of
* this software and associated documentation files (the "Software"), to deal in
* the Software without restriction, including without limitation the rights to
* use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of
* the Software, and to permit persons to whom the Software is furnished to do so,
* subject to the following conditions:
*
* The above copyright notice and this permission notice shall be included in all
* copies or substantial portions of the Software. If you wish to use our Amazon
* FreeRTOS name, please do so in a fair use way that does not cause confusion.
*
* THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
```

```
* IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY, FITNESS

* FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT
SHALL THE AUTHORS OR

* COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER

* IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN

* CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
THE SOFTWARE.

*

* http://www.FreeRTOS.org

* http://aws.amazon.com/freertos

*

* 1 tab == 4 spaces!

*/


#ifdef ARDUINO


#ifndef FREERTOS_CONFIG_H

#define FREERTOS_CONFIG_H


#include <avr/io.h>


/*-----------------------------------------------------------

* Application specific definitions.
```

```
*

* These definitions should be adjusted for your particular hardware and

* application requirements.

*

* THESE PARAMETERS ARE DESCRIBED WITHIN THE 'CONFIGURATION' SECTION OF THE

* FreeRTOS API DOCUMENTATION AVAILABLE ON THE FreeRTOS.org WEB SITE.

*

* See http://www.freertos.org/a00110.html.

*----------------------------------------------------------*/

//desativAR coisas que não presisaria na app e sem asserts: configUSE_IDLE_HOOK 1 -> 0 (nçAO MUDEI, POIS NO ARDUINO ISSO CHAMA O loop() e pdoe ser util.,

// configCHECK_FOR_STACK_OVERFLOW 1 -> 0 (não fiz ainda)

//configUSE_MALLOC_FAILED_HOOK 1 -> 0 (não fiz ainda)



// And on to the things the same no matter the AVR type...

#define configUSE_PREEMPTION                1

#define configUSE_IDLE_HOOK                 0

#define configUSE_TICK_HOOK                 0

#define configCPU_CLOCK_HZ                  ( ( uint32_t ) F_CPU ) // This F_CPU variable set by the environment

#define configMAX_PRIORITIES                4
```

```
//depurando (no STM32) deu stack overflow na task ID (Idle) - estranho... mudando configMINIMAL_STACK_SIZE de 192 -> 330 (vai aumetar todos) - arduino - voltando apra 192

#define configMINIMAL_STACK_SIZE            ( ( portSTACK_TYPE ) 192 )

#define configMAX_TASK_NAME_LEN             ( 8 )

#define configUSE_TRACE_FACILITY            0

#define configUSE_16_BIT_TICKS              1

#define configIDLE_SHOULD_YIELD             1


#define configUSE_MUTEXES                   1

//max: configUSE_RECURSIVE_MUTEXES de 1 apra 0

#define configUSE_RECURSIVE_MUTEXES         0

//max: configUSE_COUNTING_SEMAPHORES de 1 para 0

#define configUSE_COUNTING_SEMAPHORES       0

#define configUSE_QUEUE_SETS                0

#define configQUEUE_REGISTRY_SIZE           0

#define configUSE_TIME_SLICING              1

#define configCHECK_FOR_STACK_OVERFLOW      1

#define configUSE_MALLOC_FAILED_HOOK        1


#define configSUPPORT_DYNAMIC_ALLOCATION    1

#define configSUPPORT_STATIC_ALLOCATION     0

//max: adicionei configTOTAL_HEAP_SIZE

//max: ajuste para o arduino - 4k
```

```
#define configTOTAL_HEAP_SIZE      ( ( size_t ) ( 4 * 1024 ) + 100)


/* Timer definitions. */

#define configUSE_TIMERS                    1

#define configTIMER_TASK_PRIORITY           ( ( UBaseType_t ) 3 )

//max: configTIMER_QUEUE_LENGTH de 10 para 2

#define configTIMER_QUEUE_LENGTH            ( ( UBaseType_t ) 10 )

#define configTIMER_TASK_STACK_DEPTH        ( ( portSTACK_TYPE ) 85 )


/* Co-routine definitions. */

#define configUSE_CO_ROUTINES               0

#define configMAX_CO_ROUTINE_PRIORITIES     ( (UBaseType_t ) 2 )


/* Set the stack depth type to be uint16_t. */

#define configSTACK_DEPTH_TYPE              uint16_t


/* Set the stack pointer type to be uint16_t, otherwise it defaults to unsigned long */

#define portPOINTER_SIZE_TYPE               uint16_t


/* Set the following definitions to 1 to include the API function, or zero
```

```
to exclude the API function. */

//max: INCLUDE_vTaskPrioritySet 1 -> 0

#define INCLUDE_vTaskPrioritySet                1

//max: INCLUDE_uxTaskPriorityGet 1 -> 0

#define INCLUDE_uxTaskPriorityGet               1

//max: INCLUDE_vTaskDelete 1 -> 0

#define INCLUDE_vTaskDelete                     1

//max: INCLUDE_vTaskCleanUpResources 1 -> 0

#define INCLUDE_vTaskCleanUpResources           1

//max: INCLUDE_vTaskSuspend 1 -> 0

#define INCLUDE_vTaskSuspend                    1

//max: INCLUDE_vResumeFromISR 1 -> 0

#define INCLUDE_vResumeFromISR                  1

#define INCLUDE_vTaskDelayUntil                 1

#define INCLUDE_vTaskDelay                      1

#define INCLUDE_xTaskGetSchedulerState          0

#define INCLUDE_xTaskGetIdleTaskHandle          0 // create an
idle task handle.

#define INCLUDE_xTaskGetCurrentTaskHandle       0

#define INCLUDE_uxTaskGetStackHighWaterMark     1


#define configMAX(a,b)  ({ __typeof__ (a) _a = (a); __typeof__
(b) _b = (b); _a > _b ? _a : _b; })
```

```
#define configMIN(a,b)  ({ __typeof__ (a) _a = (a); __typeof__
(b) _b = (b); _a < _b ? _a : _b; })


#endif /* FREERTOS_CONFIG_H */

#endif /* ARDUINO */
```

giroscopio.c

```
          /*
* giroscopio.c
*
*  Created on: 17 de maio de 2018
*      Author: Max
*      *** Baseado em:
https://www.arduinoecia.com.br/2014/09/sensor-gy-80-
acelerometro-bussola-barometro.html
*      para o  Giroscopio L3G4200D
*      *************** Ver outra lib:
https://github.com/steamfire/WSWireLib
*      https://cadernodelaboratorio.com.br/2017/04/13/iniciando
-com-o-arduino-o-protocolo-i2c/
*
*/
#define _GIROSCOPIO_C_ARDUINO_ATIVO 1


#ifdef ARDUINO


#define _GIROSCOPIO_C_


#include "giroscopio.h"
#include "tipos.h"
```

```
#if (_GIROSCOPIO_C_ARDUINO_ATIVO == 1)

#include <Arduino.h>

extern void printlog(int qtdParams, const char *s, int valor);

extern void printlogss(const char *s, char * valor);

#define log(S) printlog(1, S, 0)

#define logval(S, I) printlog(2, S, I)

#define logvals(S, I) printlogss(S, I)

#define logdados(S, I) printlog(S, I)


extern void wire_writeRegister(int deviceAddress, byte address,
byte val);

extern int wire_readRegister(int deviceAddress, byte address);

extern void wire_begin(void);

#define byte unsigned char

typedef struct {

TickType_t xAhora;

float fValorMinimoLeituras;

 int x;
```

```
 int y;
 int z;

 struct  {
 int x;
 int y;
 int z;
 } valorParado;

} contole_giroscopio_t;

#define ABS(x)                    ( (x < 0) ? (-x) : x )

#define CTRL_REG1 0x20
#define CTRL_REG2 0x21
#define CTRL_REG3 0x22
#define CTRL_REG4 0x23
#define CTRL_REG5 0x24

static contole_giroscopio_t gxControleGiroscopio;

//Endereco I2C do L3G4200D
static int L3G4200D_Address = 105;

static void taskGerenciaGiroscopioFunc( void *pvParameters);
```

```
static void getGyroValues(contole_giroscopio_t
*pxControleGiroscopio);

static int setupL3G4200D(int scale);


#endif


/*

* @brief Inicializa o módulo

* @aoram float fValorMinimoLeituras - Valor mínimo para aceitar
a leitura

* @retval None

*/

void giroscopio_init(float fValorMinimoLeituras)

{

#if (_GIROSCOPIO_C_ARDUINO_ATIVO == 1)


contole_giroscopio_t *pxControleGiroscopio = pvPortMalloc(
sizeof(contole_giroscopio_t) );


gxControleGiroscopio.fValorMinimoLeituras =
fValorMinimoLeituras;

gxControleGiroscopio.x = 0;

gxControleGiroscopio.y = 0;

gxControleGiroscopio.z = 0;


 //configura comunicação
```

```
wire_begin();

  //Inicializando o L3G4200D
  // Configura o L3G4200 para 200, 500 ou 2000 graus/seg
  setupL3G4200D(500);

  gxControleGiroscopio.valorParado.x = 0;
  gxControleGiroscopio.valorParado.y = 0;
  gxControleGiroscopio.valorParado.z = 0;

  for(int i=0; i<500; i++)
  {
    getGyroValues(&gxControleGiroscopio);

    //logval(": ", gxControleGiroscopio.x);

    gxControleGiroscopio.valorParado.x =
(gxControleGiroscopio.valorParado.x + gxControleGiroscopio.x) /
2;
    gxControleGiroscopio.valorParado.y =
(gxControleGiroscopio.valorParado.y + gxControleGiroscopio.y) /
2;
    gxControleGiroscopio.valorParado.z =
(gxControleGiroscopio.valorParado.z + gxControleGiroscopio.z) /
2;
```

```
   //logval("media parado de x: ",
gxControleGiroscopio.valorParado.x);

 }


 //logval("media parado de x: ",
gxControleGiroscopio.valorParado.x);

 //logval("media parado de y: ",
gxControleGiroscopio.valorParado.y);

 //logval("media parado de z: ",
gxControleGiroscopio.valorParado.z);


#endif

}



#if (_GIROSCOPIO_C_ARDUINO_ATIVO == 1)



/*

* @brief Inicializa o módulo

* @param giroscopio_leitura_t &pxUltimaLeitura - Ponteiro para
a estrutura que receberá a última leitura

* @retval verdadeiro se leu, false se não foi possivel obter
uma leitura

*/

int giroscopio_lerUltimoValor(giroscopio_leitura_t
*pxUltimaLeitura)
```

```
{
static int bPrimeiraChamada = 1;


if(bPrimeiraChamada)
{
bPrimeiraChamada = 0;
vTaskDelay(1500 / portTICK_PERIOD_MS);
}


   // Atualiza os valores de X, Y e Z
   getGyroValues(&gxControleGiroscopio);



   gxControleGiroscopio.x = gxControleGiroscopio.x -
gxControleGiroscopio.valorParado.x;
   gxControleGiroscopio.y = gxControleGiroscopio.y -
gxControleGiroscopio.valorParado.y;
   gxControleGiroscopio.z = gxControleGiroscopio.z -
gxControleGiroscopio.valorParado.z;


if( (ABS(gxControleGiroscopio.x) >=
gxControleGiroscopio.fValorMinimoLeituras) ||
(ABS(gxControleGiroscopio.y) >=
gxControleGiroscopio.fValorMinimoLeituras) ||
(ABS(gxControleGiroscopio.z) >=
gxControleGiroscopio.fValorMinimoLeituras) )
{
```

```
  pxUltimaLeitura->nEixoX = gxControleGiroscopio.x;
  pxUltimaLeitura->nEixoY = gxControleGiroscopio.y;
  pxUltimaLeitura->nEixoZ = gxControleGiroscopio.z;

  return 1;
}
// else
// {
// log("[Leitura abaixo do limite mínimo!!]");
// }

return 0;


}


void getGyroValues(contole_giroscopio_t *pxControleGiroscopio)
{
  // Rotina para leitura dos valores de X, Y e Z
  byte xMSB = wire_readRegister(L3G4200D_Address, 0x29);
  byte xLSB = wire_readRegister(L3G4200D_Address, 0x28);
  pxControleGiroscopio->x = ((xMSB << 8) | xLSB);

  byte yMSB = wire_readRegister(L3G4200D_Address, 0x2B);
```

```
  byte yLSB = wire_readRegister(L3G4200D_Address, 0x2A);
  pxControleGiroscopio->y = ((yMSB << 8) | yLSB);

  byte zMSB = wire_readRegister(L3G4200D_Address, 0x2D);
  byte zLSB = wire_readRegister(L3G4200D_Address, 0x2C);
  pxControleGiroscopio->z = ((zMSB << 8) | zLSB);
}

int setupL3G4200D(int scale)
{
  //From  Jim Lindblom of Sparkfun's code

  // Enable x, y, z and turn off power down:
wire_writeRegister(L3G4200D_Address, CTRL_REG1, 0b00001111);

  // If you'd like to adjust/use the HPF, you can edit the line
below to configure CTRL_REG2:
wire_writeRegister(L3G4200D_Address, CTRL_REG2, 0b00000000);

  // Configure CTRL_REG3 to generate data ready interrupt on
INT2
  // No interrupts used on INT1, if you'd like to configure INT1
  // or INT2 otherwise, consult the datasheet:
wire_writeRegister(L3G4200D_Address, CTRL_REG3, 0b00001000);
```

```
  // CTRL_REG4 controls the full-scale range, among other
things:
  if(scale == 250){
  wire_writeRegister(L3G4200D_Address, CTRL_REG4, 0b00000000);
  }else if(scale == 500){
  wire_writeRegister(L3G4200D_Address, CTRL_REG4, 0b00010000);
  }else{
  wire_writeRegister(L3G4200D_Address, CTRL_REG4, 0b00110000);
  }

  // CTRL_REG5 controls high-pass filtering of outputs, use it
  // if you'd like:
  wire_writeRegister(L3G4200D_Address, CTRL_REG5, 0b00000000);
}

#endif

#endif
```

heap_1.c

```
          /*
             FreeRTOS V9.0.0 - Copyright (C) 2016 Real Time
Engineers Ltd.
             All rights reserved

             VISIT http://www.FreeRTOS.org TO ENSURE YOU ARE
USING THE LATEST VERSION.

             This file is part of the FreeRTOS distribution.

             FreeRTOS is free software; you can redistribute it
and/or modify it under
             the terms of the GNU General Public License
(version 2) as published by the
             Free Software Foundation >>>> AND MODIFIED BY <<<<
the FreeRTOS exception.

             ***************************************************
*************************
             >>!   NOTE: The modification to the GPL is
included to allow you to     !<<
             >>!   distribute a combined work that includes
FreeRTOS without being   !<<
             >>!   obliged to provide the source code for
proprietary components     !<<
             >>!   outside of the FreeRTOS
kernel.                                  !<<
             ***************************************************
*************************

             FreeRTOS is distributed in the hope that it will
be useful, but WITHOUT ANY
             WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS
             FOR A PARTICULAR PURPOSE.  Full license text is
available on the following
             link: http://www.freertos.org/a00114.html

             ***************************************************
*************************
              *
                       *
              *    FreeRTOS provides completely free yet
professionally developed,    *
              *    robust, strictly quality controlled,
supported, and cross           *
              *    platform software that is more than just the
market leader, it     *
```

```
             *    is the industry's de facto
standard.                                *
             *
                     *
             *    Help yourself get started quickly while
simultaneously helping     *
             *    to support the FreeRTOS project by purchasing
a FreeRTOS           *
             *    tutorial book, reference manual, or
both:                      *
             *    http://www.FreeRTOS.org/Documentation
                     *
             *
                     *
            ***********************************************
************************

            http://www.FreeRTOS.org/FAQHelp.html - Having a
problem?  Start by reading
            the FAQ page "My application does not run, what
could be wrong?".  Have you
            defined configASSERT()?

            http://www.FreeRTOS.org/support - In return for
receiving this top quality
            embedded software for free we request you assist
our global community by
            participating in the support forum.

            http://www.FreeRTOS.org/training - Investing in
training allows your team to
            be as productive as possible as early as possible.
 Now you can receive
            FreeRTOS training directly from Richard Barry, CEO
of Real Time Engineers
            Ltd, and the world's leading authority on the
world's leading RTOS.

            http://www.FreeRTOS.org/plus - A selection of
FreeRTOS ecosystem products,
            including FreeRTOS+Trace - an indispensable
productivity tool, a DOS
            compatible FAT file system, and our tiny thread
aware UDP/IP stack.

            http://www.FreeRTOS.org/labs - Where new FreeRTOS
products go to incubate.
            Come and try FreeRTOS+TCP, our new open source
TCP/IP stack for FreeRTOS.
```

```
            http://www.OpenRTOS.com - Real Time Engineers ltd.
license FreeRTOS to High
            Integrity Systems ltd. to sell under the OpenRTOS
brand.  Low cost OpenRTOS
            licenses offer ticketed support, indemnification
and commercial middleware.

            http://www.SafeRTOS.com - High Integrity Systems
also provide a safety
            engineered and independently SIL3 certified
version for use in safety and
            mission critical applications that require
provable dependability.

            1 tab == 4 spaces!
          */


          #ifdef ARDUINO

          //max: indica que este sketch tem seu próprio
alocador
          #define ALOCACAO_JA_DEFINIDA_NO_SKETCK 1

          /*
           * The simplest possible implementation of
pvPortMalloc().  Note that this
           * implementation does NOT allow allocated memory to
be freed again.
           *
           * See heap_2.c, heap_3.c and heap_4.c for
alternative implementations, and the
           * memory management pages of http://www.FreeRTOS.org
for more information.
           */
          #include <stdlib.h>

          /* Defining MPU_WRAPPERS_INCLUDED_FROM_API_FILE
prevents task.h from redefining
          all the API functions to use the MPU wrappers.  That
should only be done when
          task.h is included from an application file. */
          #define MPU_WRAPPERS_INCLUDED_FROM_API_FILE

          #include "FreeRTOSConfig.h"
          #include <Arduino_FreeRTOS.h>
          #include "task.h"

          #undef MPU_WRAPPERS_INCLUDED_FROM_API_FILE

          #if( configSUPPORT_DYNAMIC_ALLOCATION == 0 )
```

```
          #error This file must not be used if
configSUPPORT_DYNAMIC_ALLOCATION is 0
          #endif

          /* A few bytes might be lost to byte aligning the
heap start address. */
          #define configADJUSTED_HEAP_SIZE (
configTOTAL_HEAP_SIZE - portBYTE_ALIGNMENT )

          /* Allocate the memory for the heap. */
          /* Allocate the memory for the heap. */
          #if( configAPPLICATION_ALLOCATED_HEAP == 1 )
          /* The application writer has already defined the
array used for the RTOS
          heap - probably so it can be placed in a special
segment or address. */
          extern uint8_t ucHeap[ configTOTAL_HEAP_SIZE ];
          #else
          static uint8_t ucHeap[ configTOTAL_HEAP_SIZE ];
          #endif /* configAPPLICATION_ALLOCATED_HEAP */

          static size_t xNextFreeByte = ( size_t ) 0;

          /*-----------------------------------------------------
--------*/

          void *pvPortMalloc( size_t xWantedSize )
          {
          void *pvReturn = NULL;
          static uint8_t *pucAlignedHeap = NULL;

          /* Ensure that blocks are always aligned to the
required number of bytes. */
          #if( portBYTE_ALIGNMENT != 1 )
          {
          if( xWantedSize & portBYTE_ALIGNMENT_MASK )
          {
          /* Byte alignment required. */
          xWantedSize += ( portBYTE_ALIGNMENT - ( xWantedSize &
portBYTE_ALIGNMENT_MASK ) );
          }
          }
          #endif

          vTaskSuspendAll();
          {
          if( pucAlignedHeap == NULL )
          {
          /* Ensure the heap starts on a correctly aligned
boundary. */
```

```
          pucAlignedHeap = ( uint8_t * ) ( ( (
portPOINTER_SIZE_TYPE ) &ucHeap[ portBYTE_ALIGNMENT ] ) & ( ~(
( portPOINTER_SIZE_TYPE ) portBYTE_ALIGNMENT_MASK ) ) );
          }

          /* Check there is enough room left for the
allocation. */
          if( ( ( xNextFreeByte + xWantedSize ) <
configADJUSTED_HEAP_SIZE ) &&
          ( ( xNextFreeByte + xWantedSize ) > xNextFreeByte )
)/* Check for overflow. */
          {
          /* Return the next free byte then increment the index
past this
          block. */
          pvReturn = pucAlignedHeap + xNextFreeByte;
          xNextFreeByte += xWantedSize;
          }

          traceMALLOC( pvReturn, xWantedSize );
          }
          ( void ) xTaskResumeAll();

          #if( configUSE_MALLOC_FAILED_HOOK == 1 )
          {
          if( pvReturn == NULL )
          {
          extern void vApplicationMallocFailedHook( void );
          vApplicationMallocFailedHook();
          }
          }
          #endif

          return pvReturn;
          }
          /*-----------------------------------------------
--------*/

          void vPortFree( void *pv )
          {
          /* Memory cannot be freed using this scheme.  See
heap_2.c, heap_3.c and
          heap_4.c for alternative implementations, and the
memory management pages of
          http://www.FreeRTOS.org for more information. */
          ( void ) pv;

          /* Force an assert as it is invalid to call this
function. */
          configASSERT( pv == NULL );
```

```
          }
          /*-----------------------------------------------------
--------*/

          void vPortInitialiseBlocks( void )
          {
          /* Only required when static memory is not cleared.
*/
          xNextFreeByte = ( size_t ) 0;
          }
          /*-----------------------------------------------------
--------*/

          size_t xPortGetFreeHeapSize( void )
          {
          return ( configADJUSTED_HEAP_SIZE - xNextFreeByte );
          }

          #endif /* ARDUINO */
```

joystick.c

```
          /*
* joystick.c
*
*  Created on: 17 de maio de 2018
*      Author: maxback
*/

#define _JOYSTICK_C_ARDUINO_ATIVO 0

          #ifdef ARDUINO


#define _JOYSTICK_C_


#include "joystick.h"

#if (_JOYSTICK_C_ARDUINO_ATIVO == 1)

#include <Arduino.h>


static volatile TickType_t xUltimoTiempoInterrupcion = 0;
static volatile TickType_t xAhora;
```

```
//static TimerHandle_t gxTimerLeiturasADC = NULL;
static QueueHandle_t gxQueueLeituras = NULL;

static volatile joystick_leitura_t gxLeitura;

#endif

//void timerJoystickADCFunc( TimerHandle_t xTimer );
/*
* @brief Inicialzia o módulo
* @param QueueHandle_t xQueueLeituras com itens do tipo joystick_leitura_t
* @retval None
*/
void joystick_init(QueueHandle_t xQueueLeituras)
{
#if (_JOYSTICK_C_ARDUINO_ATIVO == 1)
  gxLeitura.nLeituraX = 0;
  gxLeitura.nLeituraY = 0;
  gxLeitura.xTempoEntreLeituras = 0;

  gxQueueLeituras = xQueueLeituras;

  //inicializa hw
```

```
#endif
}


#endif
```

lcd.c

```
/*
 * lcd.c
 *
 *  Created on: 4 de jun de 2018
 *      Author: Max
 */
#ifdef ARDUINO

#define _LEDS_C_

#include "lcd.h"
#include <Arduino.h>

#include <Arduino_FreeRTOS.h>

#define LCD_ATIVA_LOG 1

#if(LCD_ATIVA_LOG == 0)

#define log(S)
#define logval(S, I);
#define logvals(S, I)

#else

#ifdef ARDUINO
extern void printlog(int qtdParams, const char *s,
int valor);
extern void printlogss(const char *s, char * valor);

#define log(S) printlog(1, S, 0)
#define logval(S, I) printlog(2, S, I)
#define logvals(S, I) printlogss(S, I);
#else
#define log(S)
#define logval(S, I);
#define logvals(S, I)
#endif
#endif


extern void lcd_begin(int colunas, int linhas);
extern void lcd_print(char *texto);
extern void lcd_setCursor(int coluna, int linha);

#define byte unsigned char

#define LINHAS 2
#define COLUNAS 16

/*
```

```
           * @brief Inicializa o LCD
           * @retval none
           */
          void lcd_init(void)
          {
          lcd_begin(COLUNAS, LINHAS);
          }

          /*
           * @brief Retorna configurações de tamanho do display
           * @param int *nQtdColunas - Retorna a quantidade de
colunas do display presente - 0 se nenhum
           * @param int *nQtdLinhas  - Retorna a quantidade de
linhas do display presente - 0 se nenhum
           * @retval int - verdadeiro se tem um display
implementado na plataforma e comunciando
           */
          int lcd_getConfig(int *nQtdColunas, int *nQtdLinhas)
          {
          *nQtdColunas = COLUNAS;
          *nQtdLinhas = LINHAS;

          return 1;
          }
          /*
           * @brief define a posição nova do cursor
           * @param int nColuna - Define a coluna em que deve
posicionar (0 a n-1)
           * @param int *nLinha  - Define a linha em que deve
posicionar (0 a n-1)
           * @retval none
           */
          void lcd_definirCursor(int nColuna, int nLinha)
          {
          taskENTER_CRITICAL();

          lcd_setCursor(nColuna, nLinha);

          taskEXIT_CRITICAL();

          }

          /*
           * @brief limpa o lcd e posiciona na primeiral inha e
coluna
           * @retval none
           */
          void lcd_limpa(char cPreenchimento)
          {
          char s[2];
```

```
            taskENTER_CRITICAL();

            s[0] = cPreenchimento;
            s[1] = '\0';

            for(int l=0; l<LINHAS; l++)
            {
            for(int c=0; c<COLUNAS; c++)
            {
            lcd_print(s);
            }
            }
            lcd_setCursor(0, 0);

            taskEXIT_CRITICAL();

            }

            /*
             * @brief escreve o texto na posicao atual em diante
             * char *szTexto = Testa a ser impresso
             * int bQuebraAutomatica - true se desejar que sempre
mude de linha automaticamente,
             * false se deve simplemente enviar otexto para o lcd
             * @retval nonegdb
             */
            void lcd_texto(char *szTexto)
            {
            taskENTER_CRITICAL();

            lcd_print(szTexto);

            taskEXIT_CRITICAL();

            }

            #endif
```

leds.c

```
          /*
* leds.c
*
*  Created on: 17 de maio de 2018
*      Author: maxback
*/

          #ifdef ARDUINO

#define _LEDS_C_

#include "leds.h"
#include <Arduino.h>



static const int pinoLedVerde = 5;
static const int pinoLedAmarelo = 26;
static const int pinoLedVermelho1 = 48;
static const int pinoLedVermelho2 = 44;

static int iniciado = 0;

void leds_init(void)
{
```

```
    //config hw
  pinMode(pinoLedVerde, OUTPUT);
  pinMode(pinoLedAmarelo, OUTPUT);
  pinMode(pinoLedVermelho1, OUTPUT);
  pinMode(pinoLedVermelho2, OUTPUT);

  iniciado = 1;
}

void leds_liga(leds_selecao_led_t led)
{
if(!iniciado) return;

switch(led)
{
case LEDS_LED_TODOS:

case LEDS_LED_VERDE:
digitalWrite(pinoLedVerde, HIGH);
if(led != LEDS_LED_TODOS)
break;

case LEDS_LED_LARANJA:
digitalWrite(pinoLedAmarelo, HIGH);
```

```
if(led != LEDS_LED_TODOS)
break;

case LEDS_LED_VERMELHO:
digitalWrite(pinoLedVermelho1, HIGH);
if(led != LEDS_LED_TODOS)
break;

case LEDS_LED_AZUL:
digitalWrite(pinoLedVermelho2, HIGH);
if(led != LEDS_LED_TODOS)
break;
}

}


void leds_desliga(leds_selecao_led_t led)
{
if(!iniciado) return;

switch(led)
{
case LEDS_LED_TODOS:
```

```
case LEDS_LED_VERDE:
digitalWrite(pinoLedVerde, LOW);
if(led != LEDS_LED_TODOS)
break;

case LEDS_LED_LARANJA:
digitalWrite(pinoLedAmarelo, LOW);
if(led != LEDS_LED_TODOS)
break;

case LEDS_LED_VERMELHO:
digitalWrite(pinoLedVermelho1, LOW);
if(led != LEDS_LED_TODOS)
break;

case LEDS_LED_AZUL:
digitalWrite(pinoLedVermelho2, LOW);
if(led != LEDS_LED_TODOS)
break;
}

}

void leds_pisca(leds_selecao_led_t led)
```

```
{
if(!iniciado) return;

switch(led)
{
case LEDS_LED_TODOS:

case LEDS_LED_VERDE:
digitalWrite(pinoLedVerde, digitalRead(pinoLedVerde)==HIGH ?
LOW : HIGH);
if(led != LEDS_LED_TODOS)
break;

case LEDS_LED_LARANJA:
digitalWrite(pinoLedAmarelo, digitalRead(pinoLedAmarelo)==HIGH
? LOW : HIGH);
if(led != LEDS_LED_TODOS)
break;

case LEDS_LED_VERMELHO:
digitalWrite(pinoLedVermelho1,
digitalRead(pinoLedVermelho1)==HIGH ? LOW : HIGH);
if(led != LEDS_LED_TODOS)
break;
```

```
case LEDS_LED_AZUL:

digitalWrite(pinoLedVermelho2,
digitalRead(pinoLedVermelho2)==HIGH ? LOW : HIGH);

if(led != LEDS_LED_TODOS)

break;

}

}


#endif
```

serial.c

```
/*
 * serial.c
 *
 *  Created on: 17 de maio de 2018
 *      Author: Max
 *      Adaptado de:
 */

#ifdef ARDUINO

#define _SERIAL_C_

#include "serial.h"

#include <Arduino_FreeRTOS.h>
#include <semphr.h>
#include <timers.h>
#include <task.h>
#include <Arduino.h>

#include "tipos.h"

extern void ExecutarSerial1_begin(unsigned long
baud);
extern int ExecutarSerial1_available(void);
extern  int ExecutarSerial1_read(void);
extern  void ExecutarSerial1_print(char *s);
extern int
ExecutarSerial1_testarEstahTransmitindo(void);
extern void
ExecutarSerial1_registrarCalbackSerialEvent(callbackGenerico_t
pCallback);

extern void printlog(int qtdParams, const char *s,
int valor);

#define log(S)
//printlog(1, S, 0);
#define logval(S, I)
//printlog(2, S, I);
#define logvals(S, I)
//printlog(3, S, (int)I);


#define USA_TIMEOUT_SW 1
#define SERIAL_TIMEOUT_RX 1000

static volatile QueueHandle_t gxColaEnvio = NULL;
static volatile QueueHandle_t gxColaRecepcao = NULL;
```

```
          #if(USA_TIMEOUT_SW == 1)
            static volatile TimerHandle_t gxTimerRecepcao =
NULL;
          #endif

          static SemaphoreHandle_t SemaforoAcessoBufferRx =
NULL;


          #define SERIAL2_BUFFER_RX_SIZE 400
          static volatile size_t gnTamBufferRx = 0;

          static volatile char
gszBufferRx[SERIAL2_BUFFER_RX_SIZE];


          #if(USA_TIMEOUT_SW == 1)
          void callbackTimeoutEntreCaracteresRx( TimerHandle_t
xTimer __attribute__((unused)));
          #endif

          static void trataCallBackSerialEvent(void
*pvParametos);

          /*
           * @brief Inicialzia o módulo
           * @param nONE
           * @retval None
           */
          void serial_init(void)
          {

            //configura o hw


            ExecutarSerial1_begin(19200);
            ExecutarSerial1_registrarCalbackSerialEvent(trataCal
lBackSerialEvent);

            SemaforoAcessoBufferRx = xSemaphoreCreateMutex();

          #if(USA_TIMEOUT_SW == 1)
          //time de disparo unico para terminar Rx por timeout
          gxTimerRecepcao = xTimerCreate("TimerRecepcaoSerial",
          SERIAL_TIMEOUT_RX / portTICK_PERIOD_MS, //convete
para ticks
          pdFALSE, //sem autoreload (one shot)
          0, //timer ID
          callbackTimeoutEntreCaracteresRx );

          #endif
```

```
            return;
           }

           void trataCallBackSerialEvent(void *pvParametos)
           {
            (void *) pvParametos;

            char c;
            //acesso exclusivo via semáforo ao buffer
            xSemaphoreTake( SemaforoAcessoBufferRx,
portMAX_DELAY );

            while (ExecutarSerial1_available())
            {

           c = (char)ExecutarSerial1_read();

           if(gnTamBufferRx < sizeof(gszBufferRx) )
           {

           gszBufferRx[gnTamBufferRx++] = c;


           #if(USA_TIMEOUT_SW == 1)
           if( !(gnTamBufferRx%10) )
           {
            //log("[reiniciando timer Rx]");
            xTimerStart ( gxTimerRecepcao, 0);
           }
           #endif
           }
           else
           {
           //acesso exclusivo via semáforo ao buffer
           logval("cheio - Qtd bytes: ", gnTamBufferRx);
           serial_copiarRxParaFila();
           }

            }
            xSemaphoreGive( SemaforoAcessoBufferRx );
           }

           /*
            * @brief Configura ligacao do pressiona do botão
            * com uma fila
            * @param QueueHandle_t xCola - Fila de linhas para
envio
            * @retval None

            */
```

```
          void serial_setFilaEnvio(QueueHandle_t xCola)
          {
          gxColaEnvio = xCola;
          }

          /*
           * @brief Configura ligacao do pressiona do botão
           * com uma fila
           * @param QueueHandle_t xCola - Fila de caracteres
recebidos
           */
          void serial_setFilaRecepcao(QueueHandle_t xCola)
          {
          gxColaRecepcao = xCola;
          }


          /*
           * Pede para popular do buffer de x para a fila de
caracteres.
           * Vai fazer isso em uma seção critica
           */
          void serial_copiarRxParaFila(void)
          {
          int i;
          int iniciou = 0;
          int tentativas = 5;

          log("-------->");
          if(gnTamBufferRx == 0)
          {
          log("a");
          #if(USA_TIMEOUT_SW == 1)
          xTimerStart ( gxTimerRecepcao, 0);
          #endif
          return;
          }

          log("b");
          for(i=0; i<gnTamBufferRx; i++)
          {
          log("c");
          if(!iniciou && !gszBufferRx[i])
          {
          log("d");
          continue;
          }

          log("e");
```

```
          iniciou = 1;
          if(xQueueSendToBack(gxColaRecepcao, (void
*)&gszBufferRx[i], 1000 / portTICK_PERIOD_MS /*portMAX_DELAY*/)
== pdPASS)
          {
          log("*");
          log("f");

          tentativas = 5;

          gszBufferRx[i] = 0; //zera aquo ao inves do em um
loop

          if( i == (gnTamBufferRx-1) )
          {
          log("g");

          // manda 0 para sinalizar fim de linha na tesk que
junta em linhas
          gszBufferRx[i] = '\0';

          xQueueSendToBack(gxColaRecepcao, (void
*)&gszBufferRx[i], 0);
          log("h");

          gnTamBufferRx = 0;
          break;
          }
          }
          else
          {
          log("i");
          //eperando infilitamente não perderia, se nã
oesperar, zera contador aqui, ficando o que botou na fila
          //e perdendo o resto

          if(!tentativas--)
          {
          gnTamBufferRx = 0;

          log("J");

          break;
          }
          }
          }

          log("--- k ---");

          }

          /*
```

```
          * retorna um numero maior que 0 se estiver
trasmitindo (a qtd de bytes ainda não transmitida)
          */
          size_t serial_testarEstaTransmitindo(void)
          {
          return ExecutarSerial1_testarEstahTransmitindo();
          }

          volatile char * copiarTexto(volatile char *szDestino,
char *szOrigem, size_t nTamBufferDestino)
          {
          volatile char *szRet = szDestino;

          if(szOrigem == szDestino)
          {
          szDestino[nTamBufferDestino-1] = '\0';
          return szRet;
          }

          *szDestino = '\0';
          while(*szOrigem)
          {
          *szDestino = *szOrigem;
          szOrigem++;
          szDestino++;
          *szDestino = '\0';

          nTamBufferDestino--;

          if(nTamBufferDestino == 1)
          {
          *szDestino = '\0';
          break;
          }
          }
          return szRet;
          }

          /*
          * Define dados do buffer de envio
          */
          void serial_inicializaEnvio(char *szBuffer, size_t
nTam)
          {

          //dispaa envio
          ExecutarSerial1_print(szBuffer);
          }

          /*
```

```
           * @brief Habilitar o envio e dispara int. que inicia
transmissão
           * @param None
           */
          void serial_habilitaEnvio(void)
          {

          //nada precisa ser feito apra isso aqui
          return;
          }


          /*
           * @brief função de callback para o timeout entre
caracteres
           * @param pvParameters
           * @retval TaskHandle_t
           */
          #if(USA_TIMEOUT_SW == 1)
          void callbackTimeoutEntreCaracteresRx( TimerHandle_t
xTimer)
          {

          //de 161204_Mastering_the_FreeRTOS_Real_Time_Kernel-
A_Hands-On_Tutorial_Guide.pdf:
          /*
          * Software timer callback functions must not call
FreeRTOS API functions that will result in the
             * calling task entering the Blocked state, as to
do so will result in the daemon task entering the
             * Blocked state.
          */
          //acesso exclusivo via semáforo ao buffer
          if(xSemaphoreTake( SemaforoAcessoBufferRx, 0 ) !=
pdPASS)
          {
             logval("timeout será recarregado para tentar
depois - Qtd bytes: ", gnTamBufferRx);
          // Redispara temporização para tentar novamente em um
período
          xTimerStart ( xTimer, 0);
          return;
          }

             logval("timeout - Qtd bytes: ", gnTamBufferRx);

             serial_copiarRxParaFila();

          xSemaphoreGive( SemaforoAcessoBufferRx );
```

```
}
#endif

#endif
```

aplicacao.c

```
          #ifdef ARDUINO
          #include <Arduino_FreeRTOS.h>
          #else
          #include <FreeRTOS.h>
          #endif

          #include <queue.h>
          #include <task.h>

          #include "task_giroscopio.h"
          #include "task_serial.h"
          #include "task_saudacao.h"
          #include "task_ecoserial.h"
          #include "task_botao_usuario.h"
          #include "task_joystick.h"
          #include "task_cliente_http.h"
          #include "util_memory.h"

          #include "global.h"

          #ifdef ARDUINO
          #include <stdio.h>
          #include <string.h>
          #include "lcd.h"
          #else
          #include <plataforma/includes/lcd.h>
          #endif

          #ifdef ARDUINO
          extern void printlog(int qtdParams, const char *s,
int valor);
          #define log(S) printlog(1, S, 0);
          #define logval(S, I) printlog(2, S, I);
          #else
          #define log(S)
          #define logval(S, I);
          #endif

          #if(USAR_BOTAO == 1)
          static const char
*gszConfigMascaraPacoteBotao       = "BA:%d;";
          #endif

          #if(APLICACAO_QTD_MENSAGENS_INICIAIS > 0)
          static const char *gszConfigTextoPrimiroEnvioHttp   =
"SI:;";
          #endif
```

```
          #if (USAR_CLIHTTP == 1)
          static const char
*gszPacotePacoteBotaoUsuario       = "BA:0;"; //tem se ficar
compativel com a mascara
          #endif

          #if(USAR_GIROSCOPIO == 1)
          static const char * gszConfigMascaraPacoteGiroscopio
= "GL:%s,%s,%s;";
          static const char * gszConfigMascaraTipoFiltroGiro
  = "GC:F,%d;";
          #endif

          #if(USAR_JOYSTICK == 1)
          static const char * gszConfigMascaraPacoteJoystick
  = "JL:%s,%s;";
          #endif


          static QueueHandle_t gxColaEnvioSerial = NULL;
          static QueueHandle_t gxColaRecepcaoSerial = NULL;
          static QueueHandle_t gxColaDadosEnvioHttp = NULL;

          static void taskAplicacoCtrlFitroGiroFunc( void
*pvParameters );

          #define TAM_MAX_ENVIAO_ITEM_HTTP 13

          #if( (USAR_GIROSCOPIO == 0) && (USAR_JOYSTICK == 1) )

          #if(GLOBAL_TAMANHO_PACOTE_JOYSTICK >
TAM_MAX_ENVIAO_SERIAL)
            #undef TAM_MAX_ENVIAO_ITEM_HTTP
            #define TAM_MAX_ENVIAO_ITEM_HTTP
GLOBAL_TAMANHO_PACOTE_JOYSTICK
          #endif

          #elif (  (USAR_GIROSCOPIO == 1) && (USAR_JOYSTICK ==
0)  )

          #if(GLOBAL_TAMANHO_PACOTE_GIROSCOPIO >
TAM_MAX_ENVIAO_SERIAL)
            #undef TAM_MAX_ENVIAO_ITEM_HTTP
            #define TAM_MAX_ENVIAO_ITEM_HTTP
GLOBAL_TAMANHO_PACOTE_GIROSCOPIO
          #endif

          #elif (  (USAR_GIROSCOPIO == 1) && (USAR_JOYSTICK ==
1)  )

          #define MAIOR_GIRO_JOYSTICK
((GLOBAL_TAMANHO_PACOTE_GIROSCOPIO >
```

```
GLOBAL_TAMANHO_PACOTE_JOYSTICK) ?
GLOBAL_TAMANHO_PACOTE_GIROSCOPIO :
GLOBAL_TAMANHO_PACOTE_JOYSTICK)
          #if(MAIOR_GIRO_JOYSTICK > TAM_MAX_ENVIAO_SERIAL)
           #undef TAM_MAX_ENVIAO_ITEM_HTTP
           #define TAM_MAX_ENVIAO_ITEM_HTTP MAIOR_GIRO_JOYSTICK
          #endif
          #endif

          #if(USAR_GIROSCOPIO == 1)
          //log("Criando param. giroscopio...");
          static taskgiroscopio_paraminit_t gxParamGiroscopio;
          #endif


          #if (USAR_CLIHTTP == 1)
          BaseType_t enviatextoHttp(char *szTexto)
          {
          if(gxColaDadosEnvioHttp == NULL)
          return 0;

          utilmem_copiarTexto(szTexto, szTexto,
TAM_MAX_ENVIAO_ITEM_HTTP);
          return xQueueSendToBack(gxColaDadosEnvioHttp,
szTexto, 0) == pdPASS;
          }

          BaseType_t enviatextoHttpConst(const char *szTexto)
          {
          if(gxColaDadosEnvioHttp == NULL)
          return 0;

          return xQueueSendToBack(gxColaDadosEnvioHttp,
szTexto, 0) == pdPASS;
          }

          #endif

          void logCriacaoFila(char *szTexto, size_t nTamFila,
size_t nTamItemFilaTexto)
          {
          logval("heap size atual: ", xPortGetFreeHeapSize());
          logval(szTexto, nTamFila);
          logval("Tam. (em bytes) de cada item: ",
nTamItemFilaTexto);
          }


          //função de callback para filtro
```

```
          BaseType_t
funcaocallbackFiltroGiroscopio(giroscopio_leitura_t *xleitura)
          {
          /*
          //leituras anteriores
          static  giroscopio_leitura_t xAlterior = {0.0, 0.0,
0.0};
          double delta;

          // Calcula o delta entre a leitura atual e a atual
          delta = xleitura->nEixoX - xAlterior.nEixoX;
          xAlterior.nEixoX = xleitura->nEixoX;
          xleitura->nEixoX = delta;

          delta = xleitura->nEixoY - xAlterior.nEixoY;
          xAlterior.nEixoY = xleitura->nEixoY;
          xleitura->nEixoY = delta;

          delta = xleitura->nEixoZ - xAlterior.nEixoZ;
          xAlterior.nEixoZ = xleitura->nEixoZ;
          xleitura->nEixoZ = delta;
          */
          xleitura->nEixoX = xleitura->nEixoX < 0 ? xleitura-
>nEixoX * -1 : xleitura->nEixoX;
          xleitura->nEixoY = xleitura->nEixoY < 0 ? xleitura-
>nEixoY * -1 : xleitura->nEixoY;
          xleitura->nEixoZ = xleitura->nEixoZ < 0 ? xleitura-
>nEixoZ * -1 : xleitura->nEixoZ;

          return 1;
          }


          /* *
           * Função chamada para inicxializar o sistema com
suas configurações e
           * inicializações do módulos de alto nível e também
das taks e demais
           * coisas do freeRTOS.
           */
          void vIniciarSistema( void )
          {
          log("vIniciarSistema() - Iniciando aplicacao...");
          logval("heap size atual: ", xPortGetFreeHeapSize());


          #if(USAR_LCD == 1)
          logval("heap size atual: ", xPortGetFreeHeapSize());
          log("Inicialziando lcd...");
          lcd_init();
```

```
          lcd_limpa('_');
          lcd_texto("FreeRTOSWearable");
          lcd_texto("Controller inic.");
          #endif


          #if(USAR_JOYSTICK == 1)
          logval("heap size atual: ", xPortGetFreeHeapSize());
          log("Criando param. joystick...");
          logval("heap size atual: ", xPortGetFreeHeapSize());
          taskjoystick_paraminit_t xParamJoystick;
          #endif


          size_t nTamFila;
          size_t nTamFilaEnvPacSerial;
          size_t nTamFilaRecPacSerial;

          //para se comunicar com a task -
          nTamFilaEnvPacSerial = TAM_FILA_TX_SERIAL;
          //logCriacaoFila("Criando fila AppMainEnvPacSerial.
Qtd itens: ", nTamFilaEnvPacSerial, TAM_ITEM_FILA_TX_SERIAL);
          gxColaEnvioSerial =
xQueueCreate(nTamFilaEnvPacSerial, TAM_ITEM_FILA_TX_SERIAL);
//teste com item de TAM_ITEM_FILA_TX_SERIAL

          //logval("heap size atual: ",
xPortGetFreeHeapSize());
          //log("Registrando fila AppMainEnvPacSerial...");
             vQueueAddToRegistry( gxColaEnvioSerial,
"AppMainEnvPacSerial");

             nTamFilaRecPacSerial = TAM_FILA_RX_SERIAL;
          //logCriacaoFila("Criando fila AppMainRecPacSerial.
Qtd itens: ", nTamFilaRecPacSerial, TASKSERIAL_MAXTAMLINHA);
          gxColaRecepcaoSerial =
xQueueCreate(nTamFilaRecPacSerial, TASKSERIAL_MAXTAMLINHA);

          //logval("heap size atual: ",
xPortGetFreeHeapSize());
          //log("Registrando fila AppMainRecPacSerial...");
             vQueueAddToRegistry( gxColaRecepcaoSerial,
"AppMainRecPacSerial");

          #if (USAR_CLIHTTP == 1)
          nTamFila = TAM_FILA_DADOS_ENVIO_HTTP;
          //logCriacaoFila("Criando fila AppDadosHttp. Qtd
itens: ", nTamFila, TAM_MAX_ENVIAO_ITEM_HTTP);
             gxColaDadosEnvioHttp = xQueueCreate(nTamFila,
TAM_MAX_ENVIAO_ITEM_HTTP);
```

```
          //logval("heap size atual: ",
xPortGetFreeHeapSize());
             //log("Registrando fila AppDadosHttp...");
             vQueueAddToRegistry( gxColaDadosEnvioHttp,
"AppDadosHttp");
          #endif

          #if(USAR_SERIAL == 1)
             //logval("heap size atual: ",
xPortGetFreeHeapSize());
             //log("Criando taskserial...");

             taskserial_create(
                   gxColaEnvioSerial,
          TAM_ITEM_FILA_TX_SERIAL,  //teste com
TAM_ITEM_FILA_TX_SERIAL
          nTamFilaEnvPacSerial,

          gxColaRecepcaoSerial,
              TASKSERIAL_MAXTAMLINHA,
          nTamFilaRecPacSerial,

              4, NULL, 0);


          #endif


             logval("heap size atual: ",
xPortGetFreeHeapSize());  log("criando task ajusta filtro
giriscópio");
          xTaskCreate(taskAplicacoCtrlFitroGiroFunc,
                TASKCTRLCALIBRACAOGIROSCOPIO_NOME,
          TASKCTRLCALIBRACAOGIROSCOPIO_STACK_SIZE,
          (void *) gxColaDadosEnvioHttp,
          TASKCTRLCALIBRACAOGIROSCOPIO_PRIORITY,
          NULL);


          #if (USAR_CLIHTTP == 1)
          logval("heap size atual: ", xPortGetFreeHeapSize());
          log("taskclientehttp {");
             log(CLIENTEHTTP_REDE);
             //log(CLIENTEHTTP_SENHA);
             //log(CLIENTEHTTP_HOST);
             //log(CLIENTEHTTP_PORTA);

             char
szSubstringIgnorarItensFila[TASKCLIHTTP_TAM_BUFFER_FILTROS];

             //config para ignorar botão 0
```

```
            strcpy(szSubstringIgnorarItensFila,
gszPacotePacoteBotaoUsuario);

            log("} criando...");
          taskclientehttp_create(gxColaDadosEnvioHttp,
TAM_MAX_ENVIAO_ITEM_HTTP,
          gxColaRecepcaoSerial, gxColaEnvioSerial,
          TASKSERIAL_MAXTAMLINHA, TAM_ITEM_FILA_TX_SERIAL,
//teste com TAM_ITEM_FILA_TX_SERIAL
          //ver global.h
          CLIENTEHTTP_REDE, CLIENTEHTTP_SENHA,
CLIENTEHTTP_HOST, CLIENTEHTTP_PORTA,
          1, szSubstringIgnorarItensFila
          );

          #endif

          #if(USAR_BOTAO == 1)
          logval("heap size atual: ", xPortGetFreeHeapSize());
          log("Criando
taskbotaousuario(gxColaDadosEnvioHttp)...");
          taskbotaousuario_create(gxColaDadosEnvioHttp,
TAM_MAX_ENVIAO_ITEM_HTTP, gszConfigMascaraPacoteBotao,
          100);
          #endif

          #if(APLICACAO_QTD_MENSAGENS_INICIAIS > 0)
          //logval("heap size atual: ",
xPortGetFreeHeapSize());
          //log("Criando msg primeiro envio...");
          enviatextoHttpConst(gszConfigTextoPrimiroEnvioHttp);
          #endif


          #if (USAR_CLIHTTP == 0)
          #if(USAR_BOTAO == 1)
          //logval("heap size atual: ",
xPortGetFreeHeapSize()); log("Criando
taskbotaousuario(gxColaEnvioSerial)...");
          taskbotaousuario_create(gxColaEnvioSerial,
TAM_MAX_ENVIAO_ITEM_HTTP, 100);
          #endif
          #endif

          #if(USAR_GIROSCOPIO == 1)
          logval("heap size atual: ", xPortGetFreeHeapSize());
log("taskgiroscopio {");
          //log("definindo mascara envio...");
          gxParamGiroscopio.szMascaraPacotes =
gszConfigMascaraPacoteGiroscopio;
```

```
          //log(szMascara);

          #if (USAR_CLIHTTP == 1)
          //log("definindo fila gxColaDadosEnvioHttp...");
          gxParamGiroscopio.xColaPacotes =
gxColaDadosEnvioHttp;
          #else
          //log("definindo fila gxColaEnvioSerial...");
          gxParamGiroscopio.xColaPacotes = gxColaEnvioSerial;
          #endif
          //log("definindo xTempoEsperaEntreLeituras 3000...");
          gxParamGiroscopio.xTempoEsperaEntreLeituras = 1000 /
portTICK_PERIOD_MS; //le no driver de 1000 em 1000

          //log("definindo xTempoEntreLeituras 3000...");
          gxParamGiroscopio.xTempoEntreLeituras =
gxParamGiroscopio.xTempoEsperaEntreLeituras; //só aceita
leituras após acumular este intervalo nas leituras

          gxParamGiroscopio.xTipoFiltro        =
gtfMinimoValorAlgumEixo;
          //log("definindo fValorMinimoLeituras 5
graus/seg...");
          gxParamGiroscopio.fValorMinimoLeituras =
VALOR_MINIMO_FILTRO_GIRO;

          gxParamGiroscopio.pCallbackFiltro    =
funcaocallbackFiltroGiroscopio;

          log("} criando...");
          taskgiroscopio_create(&gxParamGiroscopio);
          #endif

          #if(USAR_JOYSTICK == 1)
          //logval("heap size atual: ",
xPortGetFreeHeapSize()); log("taskjoystick {");

          char szMascaraJoystick[GLOBAL_MAX_MASCARA_JOYSTICK];

          //log("definindo mascara envio...");

          xParamJoystick.szMascaraPacotes =
gszConfigMascaraPacoteJoystick;

          log(szMascaraJoystick);

          //log("definindo xTempoEntreLeituras 100...");
          xParamJoystick.xTempoEntreLeituras = 100 /
portTICK_PERIOD_MS;

          //log("} criando...");
          taskjoystick_create(&xParamJoystick);
```

```
          #endif


          logval("heap size atual: ", xPortGetFreeHeapSize());
          log("vIniciarSistema() - fim da funcao...");
          }

          #define mostra_tipo_filtro_lcd() lcd_definirCursor(0,
0); \
                lcd_texto("Atual:        "); \
                lcd_definirCursor(0, 1); \
          switch(gxParamGiroscopio.xTipoFiltro) \
                { \
          case gtfSemFiltro:                 lcd_texto("0-
SemFiltro     "); break; \
          case gtfMinimoValorAlgumEixo:      lcd_texto("1-
MinValAlgEixo "); break; \
          case gtfEliminaAbaixoDoMinimoValor: lcd_texto("2-
Elim.AbaixoMin"); break; \
          case gtfMinimoValorTodosEixo: lcd_texto("3-
Min.Val.TodosE"); break; \
          case gtfMinimoValorEixoX: lcd_texto("4-Minimo Valor
X"); break; \
          case gtfApenasValorEixoX: lcd_texto("5-ApenasValor X
"); break; \
          case gtfMinimoValorEixoY: lcd_texto("6-Minimo Valor
Y"); break; \
          case gtfApenasValorEixoY: lcd_texto("7-Apenas Valor
Y"); break; \
          case gtfMinimoValorEixoZ: lcd_texto("8-Minimo Valor
Z"); break; \
          case gtfApenasValorEixoZ: lcd_texto("9-Apenas Valor
Z"); break; \
          case gtfUsarFuncaoFiltro: lcd_texto("10-
UsarFunc.Filt"); break; \
                }

          #define mostra_selecione_tipo_filtro_lcd()
lcd_limpa(' '); \
                lcd_texto("Selecione tipo "); \
                lcd_texto("filtro (0..10):");

          /*
          * @brief Executa o código da task da aplicação para
tratar mensagem do botão 0 (usuário) e mudar tipo do filtro
          * @param void *pvParameters - Parametros usado neste
caso apra receber a fila de dados.
          * @retval none
```

```
          */
          static void taskAplicacoCtrlFitroGiroFunc( void
*pvParameters )
          {

          QueueHandle_t xCola = (QueueHandle_t) pvParameters;
          #if(USAR_GIROSCOPIO == 1)
          char szBufferItem[TAM_MAX_ENVIAO_ITEM_HTTP];
           static int bPrimeiraTecla = 1;

          for(;;)
          {
          if(xQueuePeek(xCola, (void *)szBufferItem,
portMAX_DELAY) == pdPASS)
          {
          int bEncontrou = 1;
          taskENTER_CRITICAL();
          if(strstr(szBufferItem, gszPacotePacoteBotaoUsuario)
== NULL)
          {
          bEncontrou = 0;
          }
          taskEXIT_CRITICAL();
          //se for o texto szSubstringIgnorarItensFila
          if(!bEncontrou)
          {
          //espera um ciclo, adiantando a mudança de tarefas
          vTaskDelay(1);

          continue;
          }

            log("************************");
               if(bPrimeiraTecla)
               {
                bPrimeiraTecla = 0;


             xQueueReceive(xCola, (void *)szBufferItem,
portMAX_DELAY);

                log("Selecione tipo filtro (0..10)");
                logval("** Atual: ",
gxParamGiroscopio.xTipoFiltro);
          #if(USAR_LCD == 1)
                mostra_selecione_tipo_filtro_lcd();

            vTaskDelay(3000 / portTICK_PERIOD_MS);

          mostra_tipo_filtro_lcd();
          #endif
```

```
            //coloca na fila indicando que alterou
                sprintf(szBufferItem,
gszConfigMascaraTipoFiltroGiro, gxParamGiroscopio.xTipoFiltro);
            xQueueSendToBack(xCola, (void *)szBufferItem, 0);

                sprintf(szBufferItem, "MN:0,2,Sel. giroscopio;",
gxParamGiroscopio.xTipoFiltro);
          xQueueSendToBack(xCola, (void *)szBufferItem, 0);

          sprintf(szBufferItem, "MN:1,2,filtro de 0 a 10;",
gxParamGiroscopio.xTipoFiltro);
          xQueueSendToBack(xCola, (void *)szBufferItem, 0);

          #if(USAR_LCD == 1)
            vTaskDelay(3000 / portTICK_PERIOD_MS);

                mostra_selecione_tipo_filtro_lcd();
          #endif
          log("************************");

                continue;
              }

          //muda tipo de filtro para o proximo tipo
          if(gxParamGiroscopio.xTipoFiltro ==
gtfUsarFuncaoFiltro)
          {
          gxParamGiroscopio.xTipoFiltro = gtfSemFiltro;
          gxParamGiroscopio.fValorMinimoLeituras = 0.0;
          }
          else
          {
          gxParamGiroscopio.xTipoFiltro =
gxParamGiroscopio.xTipoFiltro + 1;
          gxParamGiroscopio.fValorMinimoLeituras =
VALOR_MINIMO_FILTRO_GIRO;
          }

          taskgiroscopio_reconfigurar(&gxParamGiroscopio,
TIPOPARAMTASKNOTIFGIRO_TIPOFILTRO);

          #if(USAR_LCD == 1)
                mostra_tipo_filtro_lcd();
          #endif

          xQueueReceive(xCola, (void *)szBufferItem,
portMAX_DELAY);

          //coloca na fila indicando que alterou
          sprintf(szBufferItem, gszConfigMascaraTipoFiltroGiro,
gxParamGiroscopio.xTipoFiltro);
```

```
          log(szBufferItem);

          xQueueSendToBack(xCola, (void *)szBufferItem, 0);
//se timeout pois fila pode estar cheia

          #if(USAR_LCD == 1)
           vTaskDelay(3000 / portTICK_PERIOD_MS);

          mostra_selecione_tipo_filtro_lcd();
          #endif
          log("************************");

          }
          }
          #endif
          }
```

global.h

```
/*
 * global.h
 *
 *  Created on: 21 de abr de 2018
 *      Author: Max
 */

#ifndef APLICACAO_GLOBAL_GLOBAL_H_
#define APLICACAO_GLOBAL_GLOBAL_H_

#ifdef ARDUINO
#include <Arduino_FreeRTOS.h>
#include <Arduino.h>
#else
#include <FreeRTOS.h>
#endif


#define GLOBAL_CAST_STACK_SIZE
//(const uint16_t)
//(uint32_t)


#define USAR_SERIAL 1
#define USAR_SAUDACAO 0
#define USAR_ECO 0
#define USAR_BOTAO 1
#define USAR_GIROSCOPIO 1
#define USAR_JOYSTICK 0
#define USAR_CLIHTTP 1

#define USAR_LCD 1

#define BOTAOUSUARIO_ENVIA_MSG_PRESSIONAMENTO_FALSO 0
#define APLICACAO_QTD_MENSAGENS_INICIAIS 1

#define
CLIENTEHTTP_QTD_ERROS_HOST_PARA_RECONECTAR_REDE 50
#define
CLIENTEHTTP_QTD_TENTATIVAS_ENVIO_ANTES_DESCARTAR_ITEM 10

#define CLIENTEHTTP_REDE "tccmaxback"
#define CLIENTEHTTP_SENHA "12345678"
#define CLIENTEHTTP_HOST "192.168.43.180"
#define CLIENTEHTTP_PORTA "8080"

 #define CLIENTEHTTP_DOCUMENTO_GET
"/v0/logdispositivo.php?ID=100&i=0&t="
```

```
          #define CLIENTEHTTP_ENVIAR_DADOS_NA_REQUISICAO 1

          //Prioridade da task da serial (Rx usa esta e Tx uma
prioridade abaixo)
          #define TASKSERIAL_TASKRX_PRIORITY (
configMAX_PRIORITIES - 2 )
          //( configMAX_PRIORITIES - 1 )
          #define TASKSERIAL_TASKTX_PRIORITY (
configMAX_PRIORITIES - 2 )
          //Prioridade da task da serial
          #define TASKSAUDACAO_PRIORITY ( configMAX_PRIORITIES
- 2 )
          //Prioridade da task da serial
          #define TASKECOSERIAL_PRIORITY ( configMAX_PRIORITIES
- 2 )
          //Prioridade da task do acelormetro auemtnada pois
agora ela que
          //le via driver o acelerometro e não amsi de uma fila
          #define TASKGIROSCOPIO_PRIORITY (
configMAX_PRIORITIES - 2 )

          #define TASKBOTAOSUAURIO_PRIORITY (
configMAX_PRIORITIES - 1 )


          #define TASKJOYSTICK_PRIORITY ( configMAX_PRIORITIES
- 1 )

          #define TASKCLIENTEHTTP_PRIORITY (
configMAX_PRIORITIES - 1 )
          //( configMAX_PRIORITIES - 2 )

          #define TASKCTRLCALIBRACAOGIROSCOPIO_PRIORITY (
configMAX_PRIORITIES - 1 )


          #define TASKBOTAOSUAURIO_NOME "0BotoUsu"
          #define TASKCLIENTEHTTP_NOME  "1ClienteHttp"
          #define TASKGIROSCOPIO_NOME   "2AppGiroscopio"
          #define TASKSERIAL_TASTX_NOME "3SerialTx"
          #define TASKSERIAL_TASRX_NOME "4SerialRx"
          #define TASKJOYSTICK_NOME     "5AppJoystick"
          #define TASKSAUDACAO_NOME     "6AppSaudacao"
          #define TASKECOSERIAL_NOME    "7AppEcoSerial"
          #define TASKCTRLCALIBRACAOGIROSCOPIO_NOME
"8AppCrtlGiro"


          #ifdef ARDUINO
```

```
          #define VALOR_MINIMO_FILTRO_GIRO 50

          #define TAM_FILA_TX_SERIAL 3
          #define TAM_ITEM_FILA_TX_SERIAL 70


          #define TAM_FILA_RX_SERIAL 3
          #define TAM_FILA_DADOS_ENVIO_HTTP 9

          #define TASKSERIAL_TASRX_STACK_SIZE
GLOBAL_CAST_STACK_SIZE ( configMINIMAL_STACK_SIZE + 100  )
          #define TASKSERIAL_TASTX_STACK_SIZE
GLOBAL_CAST_STACK_SIZE ( configMINIMAL_STACK_SIZE  )
          #define TASKSAUDACAO_STACK_SIZE
GLOBAL_CAST_STACK_SIZE ( configMINIMAL_STACK_SIZE )
          #define TASKECOSERIAL_STACK_SIZE
GLOBAL_CAST_STACK_SIZE ( configMINIMAL_STACK_SIZE )
          #define TASKGIROSCOPIO_STACK_SIZE
GLOBAL_CAST_STACK_SIZE ( configMINIMAL_STACK_SIZE )
          #define TASKBOTAOSUAURIO_STACK_SIZE (
configMINIMAL_STACK_SIZE )
          #define TASKJOYSTICK_STACK_SIZE
GLOBAL_CAST_STACK_SIZE ( configMINIMAL_STACK_SIZE )
          #define TASKCLIENTEHTTP_STACK_SIZE
GLOBAL_CAST_STACK_SIZE ( configMINIMAL_STACK_SIZE + 300 )
          #define TASKCTRLCALIBRACAOGIROSCOPIO_STACK_SIZE (
configMINIMAL_STACK_SIZE )
          #else

          #define VALOR_MINIMO_FILTRO_GIRO 2

          #define TAM_FILA_TX_SERIAL 8
          #define TAM_ITEM_FILA_TX_SERIAL 70

          #define TAM_FILA_RX_SERIAL 8
          #define TAM_FILA_DADOS_ENVIO_HTTP 20

          #define TASKSERIAL_TASRX_STACK_SIZE
GLOBAL_CAST_STACK_SIZE ( configMINIMAL_STACK_SIZE + 16 )
          #define TASKSERIAL_TASTX_STACK_SIZE
GLOBAL_CAST_STACK_SIZE ( configMINIMAL_STACK_SIZE + 64 )
          #define TASKSAUDACAO_STACK_SIZE
GLOBAL_CAST_STACK_SIZE ( configMINIMAL_STACK_SIZE + 16 )
          #define TASKECOSERIAL_STACK_SIZE
GLOBAL_CAST_STACK_SIZE ( configMINIMAL_STACK_SIZE )
          #define TASKGIROSCOPIO_STACK_SIZE
GLOBAL_CAST_STACK_SIZE ( configMINIMAL_STACK_SIZE + 16 )
          #define TASKBOTAOSUAURIO_STACK_SIZE (
configMINIMAL_STACK_SIZE + 16 )
```

```
          #define TASKJOYSTICK_STACK_SIZE
GLOBAL_CAST_STACK_SIZE ( configMINIMAL_STACK_SIZE + 16 )
          #define TASKCLIENTEHTTP_STACK_SIZE
GLOBAL_CAST_STACK_SIZE ( configMINIMAL_STACK_SIZE + 200 )
          #define TASKCTRLCALIBRACAOGIROSCOPIO_STACK_SIZE (
configMINIMAL_STACK_SIZE )
          #endif
          /*
           * Tamanho máximo máximo da mascara
           */
          #define GLOBAL_MAX_MASCARA_GIROSCOPIO 13

          #define GLOBAL_MAX_MASCARA_JOYSTICK 10


          /*
           * Tamanho do pacote com dados do acelerometro
           */
          #define GLOBAL_TAMANHO_PACOTE_GIROSCOPIO
(GLOBAL_MAX_MASCARA_GIROSCOPIO + (3 * 6))

          #define GLOBAL_TAMANHO_PACOTE_JOYSTICK
(GLOBAL_MAX_MASCARA_JOYSTICK + (2 * 6))
          /*
           * @brief Definição blobal de função apra envio de
texto para uso em callback
           * permitindo injeçã ode depedencia em um m´´odulo
que pode entao enviar
           * coisas pela serial, sem saber realmente se quer o
módulo que realmente faz isso
           */
          typedef BaseType_t (*callbackEnvioTexto_t)(char
*szTexto);

          #define MAIOR_PACOTE_DADOS_HTTP
(GLOBAL_TAMANHO_PACOTE_GIROSCOPIO >
GLOBAL_TAMANHO_PACOTE_JOYSTICK ? \
          GLOBAL_TAMANHO_PACOTE_GIROSCOPIO :
GLOBAL_TAMANHO_PACOTE_JOYSTICK)

          #define CLIENTEHTTP_TAMANHO_ENVIO 200


          #endif /* APLICACAO_GLOBAL_GLOBAL_H_ */
```

task_botao_usuario.c

```
/*
 * task_botao_usuario.c
 *
 *  Created on: 1 de mai de 2018
 *      Author: Max
 */

#define _TASKBOTAOUSUARIO_C_

#include "task_botao_usuario.h"
#include "util_memory.h"
#ifdef ARDUINO
#include "botao.h"
#include "tipos.h"
#else
#include <plataforma/includes/botao.h>
#include <plataforma/tipos.h>
#endif

typedef struct
{
QueueHandle_t gxColaEnvioTaskBotaoSerial;
size_t gnTamItemFilaEnvioTaskBotaoSerial;
TickType_t gnTempoMinimoEntreAcionamentoTicks;
QueueHandle_t gxColaNotificacaoBotao;
const char *szMascaraPacotePressionamento;
size_t nTamPacotePressionamento;
} param_dados_botao_usuario_t;


void taskBotonUsuarioFunc( void *pvParameters );


/*
 * @brief Cria a task
 * @param QueueHandle_t xColaEnvio - Fila de onde dem
os textos para envio de dados do botao
 * @param size_t nTamItemFilaEnvio - Tamanho do item
da fila de envio
 * @param char *szMascaraPacotePressionamento -
Mascara para o pacote, com um primeiro %d que poderá conter
 * o indice do botão pressionado
 * @param uint32_t nTempoMinimoEntreAcionamentoMs -
tempo em milesegundos entre acionamento
 * @retval TaskHandle_t - Task tratamento
 */
#ifndef _TASKBOTAOUSUARIO_C_
extern
```

```
          #endif
          TaskHandle_t taskbotaousuario_create(QueueHandle_t
xColaEnvio,
          size_t nTamItemFilaEnvio,
          const char *szMascaraPacotePressionamento,
          uint32_t nTempoMinimoEntreAcionamentoMs)
          {
          param_dados_botao_usuario_t *pxParam;
          TaskHandle_t xTaskHandle = NULL;

          pxParam = pvPortMalloc(
sizeof(param_dados_botao_usuario_t) );

          pxParam->gxColaEnvioTaskBotaoSerial = xColaEnvio;
          pxParam->gnTamItemFilaEnvioTaskBotaoSerial =
nTamItemFilaEnvio;
          pxParam->gnTempoMinimoEntreAcionamentoTicks =
nTempoMinimoEntreAcionamentoMs / portTICK_PERIOD_MS;
          //Vou trabalhar com task notification, então a fila
não será usada nesta implementação
          pxParam->gxColaNotificacaoBotao = NULL;

          //pxParam->gxColaNotificacaoBotao = xQueueCreate(2,
sizeof( datos_lectura_boton_t ) );
          //vQueueAddToRegistry( pxParam-
>gxColaNotificacaoBotao, "AppBotaoUsuario");

          pxParam->nTamPacotePressionamento =
strlen(szMascaraPacotePressionamento)+5;

          //pxParam->szMascaraPacotePressionamento =
pvPortMalloc( pxParam->nTamPacotePressionamento * sizeof(char)
);
          //strcpy(pxParam->szMascaraPacotePressionamento,
szMascaraPacotePressionamento);
             //como é const. aponta para o endereço passado
          pxParam->szMascaraPacotePressionamento =
szMascaraPacotePressionamento;

          xTaskCreate(taskBotonUsuarioFunc, // The function
that implements the task.
                      TASKBOTAOSUAURIO_NOME, // Text name for
the task, just to help debugging.
          TASKBOTAOSUAURIO_STACK_SIZE, // The size (in words)
of the stack that should be created for the task.
          pxParam, // A parameter that can be passed into the
task.  Not used in this simple demo.
          TASKBOTAOSUAURIO_PRIORITY,// The priority to assign
to the task.  tskIDLE_PRIORITY (which is 0) is the lowest
priority.  configMAX_PRIORITIES - 1 is the highest priority.
```

```
          &xTaskHandle );


          botao_init();

          //optei por usar tasknotification ao invés de fila
para economizar recursos
             botao_setTaskParaNotificacoes(xTaskHandle);

          //botao_setFila(pxParam->gxColaNotificacaoBotao);

          return xTaskHandle;

          }

          void taskBotonUsuarioFunc( void *pvParameters )
          {
          union {
          uint32_t ulLeituras;
          datos_lectura_boton_t leitura[2];
          } xDados;
          BaseType_t xPegouNotificacaoTask;

          param_dados_botao_usuario_t *pxParam =
(param_dados_botao_usuario_t *) pvParameters;

          char *szMsgLeitura = pvPortMalloc( pxParam-
>nTamPacotePressionamento * sizeof(char) );

          #if(BOTAOUSUARIO_ENVIA_MSG_PRESSIONAMENTO_FALSO == 1)
          char szMsgErroLeitura[] = "{\"t\": \"msg\", \"val\":
\"Botao ID 0 - Falso acionamento\"}";
          #endif
          for (;;)
          {
          xDados.leitura[0].nFlagEventoOcurrido = 0;
          xDados.leitura[1].nFlagEventoOcurrido = 0;

          //espea com a task parada até chegar algo na fila (se
configurada)
          if(pxParam->gxColaNotificacaoBotao)
          {
          if(xQueueReceive( pxParam->gxColaNotificacaoBotao,
(void *)&xDados.leitura[0], portMAX_DELAY) != pdPASS)
          continue;
            }
            else
            {
            //espera notificação com os eventos
            xPegouNotificacaoTask = xTaskNotifyWait(
```

```
            BOTAO_LEITURA_MASCARA32_BITS_FLAGEVENTO_OCORREU,
//ulBitsToClearOnEntry
                  0xFFFFFFFF, //ulBitsToClearOnExit
                  &xDados.ulLeituras, //pulNotificationValue
          portMAX_DELAY);

            configASSERT( xPegouNotificacaoTask == pdPASS );
            }

          for(int i=0; i<2; i++)
          {
          if(!xDados.leitura[i].nFlagEventoOcurrido)
          continue;

          //se respeita o tempo
          if(xDados.leitura[i].nTiempoEntreOcurrencias >=
pxParam->gnTempoMinimoEntreAcionamentoTicks)
          {
          //proteção
          if(pxParam->gnTamItemFilaEnvioTaskBotaoSerial <
pxParam->nTamPacotePressionamento)
          {
          sprintf(szMsgLeitura,"{\"t\": \"erro\", \"val\":
\"fila botao(%d < %d)\"}",
          pxParam->gnTamItemFilaEnvioTaskBotaoSerial, pxParam-
>nTamPacotePressionamento);
          }
          else
          sprintf(szMsgLeitura, pxParam-
>szMascaraPacotePressionamento, xDados.leitura[i].nID);

          xQueueSendToBack(pxParam->gxColaEnvioTaskBotaoSerial,
szMsgLeitura, 0);
          }
          else
          {
          #if(BOTAOUSUARIO_ENVIA_MSG_PRESSIONAMENTO_FALSO == 1)
          {
          szMsgErroLeitura[35] = '0' + xDados.leitura[i].nID;
          xQueueSendToBack(gxColaEnvio, szMsgErroLeitura, 0);
          }
          #endif
          }
          } //for(int i=0...
              }

          }
```

task_botao_usuario.h

```
/*
 * task_serial.h
 *
 *  Created on: 18 de abr de 2018
 *      Author: Max
 */

#ifndef APLICACAO_TASKS_TASK_BOTAOUSUARIO_H_
#define APLICACAO_TASKS_TASK_BOTAOUSUARIO_H_

#ifdef ARDUINO
#include <Arduino_FreeRTOS.h>
#else
#include <FreeRTOS.h>
#endif

#include <queue.h>
#include <task.h>
#include "global.h"

/*
 * @brief Cria a task
 * @param QueueHandle_t xColaEnvio - Fila de onde dem
os textos para envio de dados do botao
 * @param size_t nTamItemFilaEnvio - Tamanho do item
da fila de envio
 * @param char *szMascaraPacotePressionamento -
Mascara para o pacote, com um primeiro %d que poderá conter
 * o indice do botão pressionado
 * @param uint32_t nTempoMinimoEntreAcionamentoMs -
tempo em milesegundos entre acionamento
 * @retval TaskHandle_t - Task tratamento
 */
#ifndef _TASKBOTAOUSUARIO_C_
extern
#endif
TaskHandle_t taskbotaousuario_create(QueueHandle_t
xColaEnvio,
size_t nTamItemFilaEnvio,
const char *szMascaraPacotePressionamento,
uint32_t nTempoMinimoEntreAcionamentoMs);

#endif
```

task_cliente_http.c

```
/*
 * task_cliente_http.c
 *
 *  Created on: 6 de mai de 2018
 *      Author: Max
 */

#define _TASKCLIHTTP_C_
#include <string.h>
#include "task_cliente_http.h"
#include "global.h"
#include "util_memory.h"
#include <string.h>

#ifdef ARDUINO
#include "leds.h"
#include <Arduino.h>
#else
#include <plataforma/includes/leds.h>
#endif

#include <task.h>

#define TASKHTTPCLIENT_ATIVA_LOG 0

#if(TASKHTTPCLIENT_ATIVA_LOG == 0)
#define log(S)
#define logval(S, I);
#define logvals(S, I)
#define logwifidiag(P, S) sendDataConst(P, S)

#else

#ifdef ARDUINO
extern void printlog(int qtdParams, const char *s,
int valor);
extern void printlogss(const char *s, char * valor);

#define log(S) printlog(1, S, 0)
#define logval(S, I) printlog(2, S, I)
#define logvals(S, I) printlogss(S, I);
#define logwifidiag(P, S) printlog(1, S, 0)
#else
#define log(S)
#define logval(S, I);
#define logvals(S, I)
#define logwifidiag(P, S) sendDataConst(P, S)
#endif
```

```
#endif

typedef enum {
SEM_RESPOSTA = 'T',
RECEBEU_ERRO = 'E',
RECEBEU_TEXTO = 't',
} resposta_esperapor_t;

static TaskHandle_t gxTaskClienteHttpHandle = NULL;

#define EntraSC() taskENTER_CRITICAL();
#define SaiSC() taskEXIT_CRITICAL();

typedef struct{
QueueHandle_t xCola;
//char *szBufferItem; //tira de cada fila e vou cruar
um só geral
size_t nTamMaxItem;
} dados_fila_t;


typedef struct{

struct
{
char *pcBufferRequisicao;
size_t nTamBufferRequisicao;
} requisicao;

struct {
dados_fila_t DadosEnvioHttp;
dados_fila_t RecepcaoDoModem;
dados_fila_t TranmissaoParaModem;
char *szBufferItem;
} filas;

struct {
char *szNomeRedeWifi;
char *szSenhaRedeWifi;
char *szIPDestino;
char *szPortaDestino;
int bModomoduloConfigurado;

} http;

struct {
int bAchouTexto;
int nIndiceSubStrEsperaPor;

int bAchouErro;
int bSemResposta;
} resposta;
```

```
        struct {
        int nQtdFiltros;
        char filtros[TASKCLIHTTP_TAM_BUFFER_FILTROS];
        } filtroEliminacaoFilaEntrada;

        } param_dados_task_http_t;



        /*
         * @brief função de loop da task de eco da serial
         * @param pvParameters
         * @retval TaskHandle_t
         */
        static void taskClienteHttpFunc( void *pvParameters
);

        /*
         * @brief Cria a task
         * @param QueueHandle_t xColaDadosEnvioHttp - Estes
são os dados a serem enviados
         * @param size_t nTamMaxBufferDadosEnvioHttp -
tamanho do buffer com o item da fila a ser enviado
         * @param QueueHandle_t xColaRecepcaoLinha - Fila de
onde vem os textos a nível de linha
         * @param QueueHandle_t xColaEnvioLinha - fila para
enviar texto (da task_serial, por exemplo)
         * @param size_t nTamMaxBufferRecepcao - Tamamnho do
buffer de recepçao
         * @param size_t nTamMaxBufferEnvio - Indica o limite
de buffer (usado apra garantir nulo no final
         * @param char *szNomeRedeWifi
         * @param char *szSenhaRedeWifi
         * @param char *szIPDestino
         * @param char *szPortaDestino
         * @param int nQtdFiltrosEliminacaoFilaEntrada -
indica qunatos filtros de substring existem apra eliminar (ou
seja,
         * deixar de pegar o item da fila de entrada e
esperar que outra task pegue),
         * @param char *pszFiltrosEliminacaoFilaEntrada -
Ponteiro apra buffer com TASKCLIHTTP_TAM_BUFFER_FILTROS
elementos,
         * sendo o caracter nulo o divbisor de um filtro para
outro. Os filtros serão procurados como substring no
         * valor obtido da fila, se achar não enviará e
alguma outra task precisa retirá-lo.
         */
        #ifndef _TASKCLIHTTP_C_
```

```
          extern
          #endif
          TaskHandle_t taskclientehttp_create(QueueHandle_t
xColaDadosEnvioHttp,
          size_t nTamMaxBufferDadosEnvioHttp,
          QueueHandle_t xColaRecepcaoLinha, QueueHandle_t
xColaEnvioLinha,
          size_t nTamMaxBufferRecepcao, size_t
nTamMaxBufferEnvio, char *szNomeRedeWifi,
          char *szSenhaRedeWifi, char *szIPDestino, char
*szPortaDestino,
          int nQtdFiltrosEliminacaoFilaEntrada, char
*pszFiltrosEliminacaoFilaEntrada)
          {
          size_t nTamMaxBufferCompatilhado;
          param_dados_task_http_t *pxParam;

          log("taskclientehttp_create() - iniciando... ");

          leds_init();

          leds_liga(LEDS_LED_TODOS);

          pxParam = pvPortMalloc(
sizeof(param_dados_task_http_t) );
          pxParam->requisicao.nTamBufferRequisicao =
CLIENTEHTTP_TAMANHO_ENVIO;
          logval("**** nTamBufferRequisicao:", pxParam-
>requisicao.nTamBufferRequisicao);
          pxParam->requisicao.pcBufferRequisicao =
pvPortMalloc( pxParam->requisicao.nTamBufferRequisicao *
sizeof(char) );

          pxParam->http.szIPDestino = pvPortMalloc(
(strlen(szIPDestino)+1) * sizeof(char) );
          strcpy(pxParam->http.szIPDestino, szIPDestino);

          pxParam->http.szNomeRedeWifi = pvPortMalloc(
(strlen(szNomeRedeWifi)+1) * sizeof(char) );
          strcpy(pxParam->http.szNomeRedeWifi, szNomeRedeWifi);

          pxParam->http.szPortaDestino = pvPortMalloc(
(strlen(szPortaDestino)+1) * sizeof(char) );
          strcpy(pxParam->http.szPortaDestino, szPortaDestino);

          pxParam->http.szSenhaRedeWifi = pvPortMalloc(
(strlen(szSenhaRedeWifi)+1) * sizeof(char) );
          strcpy(pxParam->http.szSenhaRedeWifi,
szSenhaRedeWifi);

          pxParam->http.bModomoduloConfigurado = 0;
```

```
          //cria um buffer de item apra as tres filas com o
tamanho do maio item
          nTamMaxBufferCompatilhado =
nTamMaxBufferDadosEnvioHttp;
          if(nTamMaxBufferRecepcao > nTamMaxBufferCompatilhado)
nTamMaxBufferCompatilhado = nTamMaxBufferRecepcao;
          if(nTamMaxBufferEnvio > nTamMaxBufferCompatilhado)
nTamMaxBufferCompatilhado = nTamMaxBufferEnvio;

          pxParam->filas.szBufferItem =
pvPortMalloc(nTamMaxBufferCompatilhado);

          pxParam->filas.DatosEnvioHttp.xCola =
xColaDadosEnvioHttp;
          pxParam->filas.DatosEnvioHttp.nTamMaxItem =
nTamMaxBufferDadosEnvioHttp;

          pxParam->filas.RecepcaoDoModem.xCola =
xColaRecepcaoLinha;
          pxParam->filas.RecepcaoDoModem.nTamMaxItem =
nTamMaxBufferRecepcao;

          pxParam->filas.TranmissaoParaModem.xCola =
xColaEnvioLinha;
          pxParam->filas.TranmissaoParaModem.nTamMaxItem =
nTamMaxBufferEnvio;

          pxParam->filtroEliminacaoFilaEntrada.nQtdFiltros =
nQtdFiltrosEliminacaoFilaEntrada;
          memcpy(pxParam->filtroEliminacaoFilaEntrada.filtros,
pszFiltrosEliminacaoFilaEntrada,
          TASKCLIHTTP_TAM_BUFFER_FILTROS);



          logval("heap size atual: ", xPortGetFreeHeapSize());
          log("Criando task ClienteHttp...");

          xTaskCreate(taskClienteHttpFunc,
          TASKCLIENTEHTTP_NOME,
          TASKCLIENTEHTTP_STACK_SIZE,
          (void *)pxParam,
          TASKCLIENTEHTTP_PRIORITY,
          &gxTaskClienteHttpHandle );

          log("taskclientehttp_create() - fim da funcao...");

          return gxTaskClienteHttpHandle;


          }
```

```
          static void  sendData(volatile
param_dados_task_http_t *pxParam, char *szCommando)
          {
          size_t tamRestante = 0;
          char *p = NULL;

          if(utilmem_comprimentoTextoSecaoCritica(szCommando) <
pxParam->filas.TranmissaoParaModem.nTamMaxItem)
          {
          xQueueSendToBack(pxParam-
>filas.TranmissaoParaModem.xCola, (void *) szCommando,
portMAX_DELAY);
          }
          else
          while(*szCommando)
          {
          tamRestante = pxParam-
>filas.TranmissaoParaModem.nTamMaxItem-1;
          p = pxParam->filas.szBufferItem;
          while(tamRestante && *szCommando)
          {
          *p++ = *szCommando++;
          tamRestante--;
          }
          *p = '\0';

          xQueueSendToBack(pxParam-
>filas.TranmissaoParaModem.xCola, (void *) pxParam-
>filas.szBufferItem, portMAX_DELAY);

          }

          }

          static void  sendDataConst(volatile
param_dados_task_http_t *pxParam, const char *szCommando)
          {
          size_t tamRestante = 0;
          char *p = NULL;

          if(utilmem_comprimentoTextoSecaoCriticaConst(szComman
do) < pxParam->filas.TranmissaoParaModem.nTamMaxItem)
          {
          xQueueSendToBack(pxParam-
>filas.TranmissaoParaModem.xCola, (void *) szCommando,
portMAX_DELAY);
          }
          else
          while(*szCommando)
          {
```

```
          tamRestante = pxParam-
>filas.TranmissaoParaModem.nTamMaxItem-1;
          p = pxParam->filas.szBufferItem;
          while(tamRestante && *szCommando)
          {
          *p++ = *szCommando++;
          tamRestante--;
          }
          *p = '\0';

          xQueueSendToBack(pxParam-
>filas.TranmissaoParaModem.xCola, (void *) pxParam-
>filas.szBufferItem, portMAX_DELAY);

          }

          }


          /*
          static void  sendData(const char *szCommando)
          {
          xQueueSendToBack(gxColaEnvio,
          (void *) utilmem_copiarTextoConst(gszBufferTx,
szCommando, gnTamMaxBufferEnvio), portMAX_DELAY);
          }
          */

          static char testa_temStr(char *s, const char
*sbustring)
          {
           char ret = 0;
           if(sbustring == NULL)
          return 0;
           if(*sbustring == '\0')
          return 0;

          taskENTER_CRITICAL();

           ret = strstr(s, sbustring) != NULL;

          taskEXIT_CRITICAL();

           return ret;
          }


          #define max(a, b) (a>b?a:b)

          static resposta_esperapor_t esperaPor(volatile
param_dados_task_http_t *pxParam,
```

```
          const char *espera, const char *esperaAlternativa,
const char *textoRespErro, const TickType_t timeout, const int
tentativas)
          {
           char *szBufferResp = pxParam->filas.szBufferItem;

           int tentativasRestantes = tentativas;

           char szBufferTesteSubstring[pxParam-
>filas.RecepcaoDoModem.nTamMaxItem * 2];
           szBufferTesteSubstring[0] = '\0';

           pxParam->resposta.bSemResposta = 1;
           pxParam->resposta.nIndiceSubStrEsperaPor = -1;
           pxParam->resposta.bAchouErro = 0;
           pxParam->resposta.bAchouTexto = 0;


           while (tentativasRestantes)
           {

            if (xQueueReceive(pxParam-
>filas.RecepcaoDoModem.xCola, (void *)szBufferResp, timeout )
== pdPASS)
            {
             szBufferResp[pxParam-
>filas.RecepcaoDoModem.nTamMaxItem] = '\0';

             //logvals("Recebido: ", szBufferResp);

          taskENTER_CRITICAL();
              strcat(szBufferTesteSubstring, szBufferResp);
          taskEXIT_CRITICAL();

              tentativasRestantes = tentativas;

              if( testa_temStr(szBufferTesteSubstring, espera))
              {
             pxParam->resposta.bSemResposta = 0;
             pxParam->resposta.bAchouTexto = 1;
             pxParam->resposta.nIndiceSubStrEsperaPor = 0;

               return RECEBEU_TEXTO;
              }

              if(testa_temStr(szBufferTesteSubstring,
esperaAlternativa) )
              {
               pxParam->resposta.bSemResposta = 0;
               pxParam->resposta.bAchouTexto = 1;
               pxParam->resposta.nIndiceSubStrEsperaPor = 1;
```

```
              return RECEBEU_TEXTO;
            }

            if(testa_temStr(szBufferTesteSubstring,
textoRespErro))
            {
              pxParam->resposta.bSemResposta = 0;
              pxParam->resposta.bAchouErro = 1;
              pxParam->resposta.nIndiceSubStrEsperaPor = 2;

                 return RECEBEU_ERRO;
            }


           //copia conteudo atual para o anterior, antes de
receber novo
           utilmem_copiaMemoriaSecaoCritica(szBufferTesteSubst
ring, szBufferResp, pxParam-
>filas.RecepcaoDoModem.nTamMaxItem);
           szBufferTesteSubstring[pxParam-
>filas.RecepcaoDoModem.nTamMaxItem] = '\0';


            }
            else
             tentativasRestantes--;
           }
          return SEM_RESPOSTA;
          }

          static resposta_esperapor_t esperaPorLista(volatile
param_dados_task_http_t *pxParam,
          const char *listaEsperados[], const int
nQtdEsperados, const char *textoRespErro, const TickType_t
timeout, const int tentativas)
          {
           char *szBufferResp = pxParam->filas.szBufferItem;
           int i;
           int tentativasRestantes = tentativas;
           char szBufferTesteSubstring[pxParam-
>filas.RecepcaoDoModem.nTamMaxItem * 2];
           szBufferTesteSubstring[0] = '\0';

           pxParam->resposta.bSemResposta = 1;
           pxParam->resposta.nIndiceSubStrEsperaPor = -1;
           pxParam->resposta.bAchouErro = 0;
           pxParam->resposta.bAchouTexto = 0;

           while (tentativasRestantes)
           {
```

```
            if (xQueueReceive(pxParam-
>filas.RecepcaoDoModem.xCola, (void *)szBufferResp, timeout )
== pdPASS)
            {
             szBufferResp[pxParam-
>filas.RecepcaoDoModem.nTamMaxItem] = '\0';
          taskENTER_CRITICAL();
              strcat(szBufferTesteSubstring, szBufferResp);
          taskEXIT_CRITICAL();

              tentativasRestantes = tentativas;

              for(i=0; i<nQtdEsperados; i++)
              {
            if( testa_temStr(szBufferTesteSubstring,
listaEsperados[i]))
            {
          pxParam->resposta.bSemResposta = 0;
          pxParam->resposta.bAchouTexto = 1;
          pxParam->resposta.nIndiceSubStrEsperaPor = i;

          return RECEBEU_TEXTO;
            }
              }

              if(testa_temStr(szBufferTesteSubstring,
textoRespErro))
              {
               pxParam->resposta.bSemResposta = 0;
               pxParam->resposta.bAchouErro = 1;
               pxParam->resposta.nIndiceSubStrEsperaPor =
nQtdEsperados;

          return RECEBEU_ERRO;
              }

             //copia conteudo atual para o anterior, antes de
receber novo
             utilmem_copiaMemoriaSecaoCritica(szBufferTesteSubs
tring, szBufferResp, pxParam-
>filas.RecepcaoDoModem.nTamMaxItem);
             szBufferTesteSubstring[pxParam-
>filas.RecepcaoDoModem.nTamMaxItem] = '\0';


            }
            else
             tentativasRestantes--;
            }
          return SEM_RESPOSTA;
```

```
          }

          #define eh_caracter_normal(l)  ( (l >= 'a' && l <=
'z') || (l >= 'A' && l <= 'Z') || (l >= '0' && l <= '9') || (l
== '{') || (l == '}') || (l == '"') || (l == ':'))

          int concatenaValorHexa(char *pcDataStr, char c)
          {
          char val;
          int i;
          char *p = pcDataStr;

          for(i=4; i>=0; i -= 4, p++)
          {
          val = (c>>i)&0x0f;
          if( (val > 9) )
          {
          *p = 'A' + val - 0x0a;
          }
          else
          {
          *p = '0' + val;
          }
          }

          return 2;
          }

          #define
testaTextosCabemNoItemDaFilaTxModem(PTR_PARAM, T1, T2, T3)
(PTR_PARAM->filas.TranmissaoParaModem.nTamMaxItem  > \
          ( \
          utilmem_comprimentoTextoSecaoCritica(T1) + \
          utilmem_comprimentoTextoSecaoCritica(T2) + \
          ( T3==NULL ? 0 :
utilmem_comprimentoTextoSecaoCriticaConst(T3) )  \
                   ) \
                  )

          static char conecta(volatile param_dados_task_http_t
*pxParam)
          {

          const TickType_t timeout = 1000 / portTICK_PERIOD_MS;
          const int tentivasEsperaResp = 2;
          int novaTentativa = 0;
          //volatile char modomoduloConfigurado = 0;
          volatile int testativasEnvio = 2;
```

```
          EntraSC();
          leds_desliga(LEDS_LED_VERDE);
          SaiSC();

          while(testativasEnvio--)
          {
          xQueueReset(pxParam-
>filas.TranmissaoParaModem.xCola);
          sendDataConst(pxParam, "cmd:reset_rx"); //comando
especial para task limpar tudo
          EntraSC(); leds_pisca(LEDS_LED_LARANJA);  SaiSC();

          sendDataConst(pxParam, "AT+RST\r\n");
          if (esperaPor(pxParam, "ready", "", "ERROR", timeout,
tentivasEsperaResp) != RECEBEU_TEXTO)
          continue;

          if(!pxParam->http.bModomoduloConfigurado)
          {
          sendDataConst(pxParam, "cmd:reset_rx"); //comando
especial para task limpar tudo
          EntraSC(); leds_pisca(LEDS_LED_LARANJA);  SaiSC();

          sendDataConst(pxParam, "AT+CWMODE=3\r\n");
          if (esperaPor(pxParam, "OK", "", "ERROR", timeout,
tentivasEsperaResp)
          != RECEBEU_TEXTO)
          continue;

          sendDataConst(pxParam, "cmd:reset_rx"); //comando
especial para task limpar tudo
          EntraSC(); leds_pisca(LEDS_LED_LARANJA);  SaiSC();

          sendDataConst(pxParam, "AT+RST\r\n");
          if (esperaPor(pxParam, "ready", "", "ERROR", timeout,
          tentivasEsperaResp) != RECEBEU_TEXTO)
          continue;

          //AT+CIPMUX=0 - modo de conexão simples
          sendDataConst(pxParam, "cmd:reset_rx"); //comando
especial para task limpar tudo
          EntraSC(); leds_pisca(LEDS_LED_LARANJA);  SaiSC();

          sendDataConst(pxParam, "AT+CIPMUX=0\r\n");
          if (esperaPor(pxParam, "OK", "", "ERROR", timeout,
tentivasEsperaResp)
          != RECEBEU_TEXTO)
          continue;

          pxParam->http.bModomoduloConfigurado = 1;
          }
```

```
          //checa tamanho
          if( !testaTextosCabemNoItemDaFilaTxModem(pxParam,
"AT+CWJAP=\"\",\"\"\r\n", (pxParam->http.szNomeRedeWifi),
pxParam->http.szSenhaRedeWifi) )
          {
          continue;
          }


                sendDataConst(pxParam, "cmd:reset_rx");
//comando especial para task limpar tudo
          EntraSC(); leds_pisca(LEDS_LED_LARANJA);  SaiSC();

          taskENTER_CRITICAL();

          sprintf(pxParam->filas.szBufferItem,
"AT+CWJAP=\"%s\",\"%s\"\r\n",
          pxParam->http.szNomeRedeWifi, pxParam-
>http.szSenhaRedeWifi);
          taskEXIT_CRITICAL();

          xQueueSendToBack(pxParam-
>filas.TranmissaoParaModem.xCola, (void *)pxParam-
>filas.szBufferItem, portMAX_DELAY);

          if (esperaPor(pxParam, "WIFI CONNECTED", "", "FAIL",
timeout,
          2 * tentivasEsperaResp) != RECEBEU_TEXTO) //retorna
WIFI CONNECTED e não OK
          continue;

          //ESPERA UM POUCO APÓS RESP. QUE CONECTOU
          vTaskDelay(5000 / portTICK_PERIOD_MS);


          int envia = 4;
          novaTentativa = 0;
          const char *retOkPegaIP[] = {"APIP", "APMAC",
"STAIP", "STAMAC", "busy p..."};

          while(envia)
          {
          // pega o endereco IP (para ver se tudo ok) - não
verifico
          sendDataConst(pxParam, "cmd:reset_rx"); //comando
especial para task limpar tudo
          EntraSC(); leds_pisca(LEDS_LED_LARANJA);  SaiSC();
          sendDataConst(pxParam, "AT+CIFSR\r\n");
          envia = 0;

          if (esperaPorLista(pxParam, retOkPegaIP, 5, "ERROR",
timeout, tentivasEsperaResp) != RECEBEU_TEXTO)
```

```
          {
          sendDataConst(pxParam, "AT+CWQAP\r\n"); //DISCONECTA
do AP
          vTaskDelay(200 / portTICK_PERIOD_MS);
          novaTentativa = 1;
          break;
          }
          else if(pxParam->resposta.nIndiceSubStrEsperaPor ==
4) //4 = "busy p..."
          {
          vTaskDelay(1000 / portTICK_PERIOD_MS);
          novaTentativa = 1;
          envia--;
          }
          else
          {
          novaTentativa = 0;
          break;
          }
          }

          if(novaTentativa)
          continue;

          while(xQueueReceive(pxParam-
>filas.RecepcaoDoModem.xCola,(void *)pxParam-
>filas.szBufferItem, 0) == pdPASS)
          {
          vTaskDelay(200 / portTICK_PERIOD_MS);
          }

          EntraSC();
          leds_liga(LEDS_LED_VERDE);
          SaiSC();

          return 1;
          }


          EntraSC();
          leds_desliga(LEDS_LED_VERDE);
          SaiSC();

          return 0;
          }


          void montarDadosRequisicaoSecaoCritica(volatile
param_dados_task_http_t *pxParam)
          {
          #if (CLIENTEHTTP_ENVIAR_DADOS_NA_REQUISICAO == 1)
```

```
          char *pcBufferDadosEnvioHttp= pxParam-
>filas.szBufferItem;
          #endif

          char *pcDataStr = pxParam-
>requisicao.pcBufferRequisicao;

          taskENTER_CRITICAL();

          *pcDataStr = '\0';


          strcat(pcDataStr, "GET ");
          strcat(pcDataStr, CLIENTEHTTP_DOCUMENTO_GET);

          pcDataStr += strlen(pcDataStr);

          #if (CLIENTEHTTP_ENVIAR_DADOS_NA_REQUISICAO == 1)
           while(
          *pcBufferDadosEnvioHttp &&
          ( strlen(pxParam->requisicao.pcBufferRequisicao) <
(pxParam->requisicao.nTamBufferRequisicao-1) )
            )
           {
          if(eh_caracter_normal(*pcBufferDadosEnvioHttp))
           *pcDataStr++ = *pcBufferDadosEnvioHttp;
          else if(*pcBufferDadosEnvioHttp == ' ')
          {
           *pcDataStr++ = '+';
          }
          else
          {
           *pcDataStr++ = '%';

          pcDataStr += concatenaValorHexa(pcDataStr,
*pcBufferDadosEnvioHttp);

          }

          *pcDataStr = 0;
          pcBufferDadosEnvioHttp++;
           }
          #endif

           *pcDataStr = 0;

           strcat(pxParam->requisicao.pcBufferRequisicao, "
HTTP/1.1\r\nHost: ");
           strcat(pxParam->requisicao.pcBufferRequisicao,
pxParam->http.szIPDestino);
           strcat(pxParam->requisicao.pcBufferRequisicao,
"\r\n\r\n");
```

```
          taskEXIT_CRITICAL();

          }



          static void taskClienteHttpFunc( void *pvParameters )
          {
          const TickType_t timeout = 1000 / portTICK_PERIOD_MS;
          const TickType_t timeout_pisca_led = 1000 /
portTICK_PERIOD_MS;
          const int tentivasEsperaResp = 2;
          const int tentivasEsperaRespConexaoHost = 10;

          volatile param_dados_task_http_t *pxParam =
pvPortMalloc( sizeof(param_dados_task_http_t) );

          volatile char conectado = 0;
          volatile int testativasEnvio, i;

          volatile char szTam[10];

          volatile int QtdErrosComunicacaoHost = 0;

          utilmem_copiaMemoriaSecaoCritica((void *)pxParam,
pvParameters, sizeof(param_dados_task_http_t));

          szTam[0] = '\0';

          if(!pxParam->filas.DatosEnvioHttp.xCola || !pxParam-
>filas.TranmissaoParaModem.xCola ||
          !pxParam->filas.RecepcaoDoModem.xCola)
          {
          for(;;)
           vTaskDelay(1);
          }


          for(i=5; i; i--)
          {
          vTaskDelay(timeout_pisca_led);
          EntraSC();
          leds_pisca(LEDS_LED_TODOS);
          SaiSC();
          }

          EntraSC();
          leds_desliga(LEDS_LED_TODOS);
          leds_liga(LEDS_LED_VERMELHO);
          SaiSC();
```

```
for(;;)
{
EntraSC();

if(!conectado)
{
 leds_desliga(LEDS_LED_VERDE);
}

leds_desliga(LEDS_LED_AZUL);
leds_desliga(LEDS_LED_LARANJA);

SaiSC();


if(!conectado)
{
//se conectar já fica zerado
QtdErrosComunicacaoHost = 0;

log("Conectando com a rede...");
conectado = conecta(pxParam);
logval("Resultado: ", conectado);
}


//
if(!conectado)
{
for(i=3; i; i--)
{
EntraSC();
leds_pisca(LEDS_LED_VERMELHO);
SaiSC();
vTaskDelay(333 / portTICK_PERIOD_MS);
}

EntraSC();
leds_liga(LEDS_LED_VERMELHO);
SaiSC();

continue;
}


     //apenas le o item, sem retirar. Quando
comunicar com sucesso, retira
if(xQueuePeek(pxParam->filas.DatosEnvioHttp.xCola,
(void *)pxParam->filas.szBufferItem, 1000 /
portTICK_PERIOD_MS )
!= pdPASS)
```

```
          {

          EntraSC();
          leds_pisca(LEDS_LED_VERMELHO);
          SaiSC();

          vTaskDelay(1);
          continue;
          }

          EntraSC();
          leds_liga(LEDS_LED_VERMELHO);
          SaiSC();

          /////////////// Verifica se pega a mensagem
/////////
          if(pxParam->filtroEliminacaoFilaEntrada.nQtdFiltros)
          {
          taskENTER_CRITICAL();
          char *pstrEliminacao = pxParam-
>filtroEliminacaoFilaEntrada.filtros;
          int bIgnorado = 0;
          char bEncontroleSubStr = *pstrEliminacao;
          int i=0;

          for(int nIndice = 0; nIndice < pxParam-
>filtroEliminacaoFilaEntrada.nQtdFiltros; nIndice++)
          {
          if(!bEncontroleSubStr)
          break;

          if(strstr(pxParam->filas.szBufferItem,
pstrEliminacao) != NULL)
          {
          bIgnorado = 1;
          break;
          }

          while(i<TASKCLIHTTP_TAM_BUFFER_FILTROS)
          {
          if(*pstrEliminacao == '\0')
          {
          pstrEliminacao++;
          bEncontroleSubStr = (*pstrEliminacao) &&
(i<TASKCLIHTTP_TAM_BUFFER_FILTROS);
          break;
          }
          pstrEliminacao++;
          i++;
```

```
          }
          }
          taskEXIT_CRITICAL();

          if(bIgnorado)
          {
          log("***** Informacao ignorada para envio http:");
          log(pxParam->filas.szBufferItem);

          //espera um ciclo, adiantando a mudança de tarefas
          vTaskDelay(1);

          continue;
          }
          }

          /////////////// envia para host ///////////////

          montarDadosRequisicaoSecaoCritica(pxParam);

          taskENTER_CRITICAL();
          sprintf(szTam, "%d", strlen(pxParam-
>requisicao.pcBufferRequisicao));
          taskEXIT_CRITICAL();


          for(testativasEnvio =
CLIENTEHTTP_QTD_TENTATIVAS_ENVIO_ANTES_DESCARTAR_ITEM;
testativasEnvio; testativasEnvio--)
          {


          vTaskDelay(1);


          if(conectado && (testativasEnvio <
CLIENTEHTTP_QTD_TENTATIVAS_ENVIO_ANTES_DESCARTAR_ITEM))
          {

          for(i=4; i; i--)
          {
          EntraSC();
          if(i==4)
          {
          leds_desliga(LEDS_LED_AZUL);
          leds_desliga(LEDS_LED_LARANJA);
          }
          else
          {
          leds_pisca(LEDS_LED_AZUL);
          leds_pisca(LEDS_LED_LARANJA);
          }
```

```
          SaiSC();
          vTaskDelay(333 / portTICK_PERIOD_MS);
          }

          }

          if(!conectado)
          {
          EntraSC();
                   leds_desliga(LEDS_LED_VERDE);
                   SaiSC();

                  conectado = conecta(pxParam);

                  if(conectado)
                  {
              EntraSC();
                     leds_liga(LEDS_LED_VERDE);
                     SaiSC();

                      testativasEnvio =
CLIENTEHTTP_QTD_TENTATIVAS_ENVIO_ANTES_DESCARTAR_ITEM;
                      QtdErrosComunicacaoHost = 0;
                  }
                  else
                  {
            vTaskDelay(1);
            continue;
                  }
          }


          EntraSC(); leds_pisca(LEDS_LED_LARANJA);  SaiSC();

          xQueueReset(pxParam-
>filas.TranmissaoParaModem.xCola);
          sendDataConst(pxParam, "cmd:reset_rx"); //comando
especial para task limpar tudo


          ////////////////////////////////// Conexão com o
host ////////////////////////////////

          //verifica tamanho
          if( !testaTextosCabemNoItemDaFilaTxModem(pxParam,
"AT+CIPSTART=\"TCP\",\"%s\",%s\r\n", pxParam->http.szIPDestino,
pxParam->http.szPortaDestino) )
          {
          continue;
          }
```

```
          sendDataConst(pxParam, "cmd:reset_rx"); //comando
especial para task limpar tudo

          taskENTER_CRITICAL();
          sprintf(pxParam->filas.szBufferItem,
"AT+CIPSTART=\"TCP\",\"%s\",%s\r\n",
          pxParam->http.szIPDestino, pxParam-
>http.szPortaDestino);
          taskEXIT_CRITICAL();
          xQueueSendToBack(pxParam-
>filas.TranmissaoParaModem.xCola, (void *)pxParam-
>filas.szBufferItem, portMAX_DELAY);

          char *listaRespConexaoHost[] = {"OK", "ALREADY
CONNECTED", "CONNECT", "CLOSED"};
             esperaPorLista(pxParam, listaRespConexaoHost, 4,
"ERROR", timeout, tentivasEsperaRespConexaoHost);


             // só para debug:

             //manda para a a palca mesmo o texto apra debug
             if(pxParam->resposta.bAchouTexto)
             {
             logwifidiag(pxParam, "*** achou: ");
             sendData(pxParam, listaRespConexaoHost[pxParam-
>resposta.nIndiceSubStrEsperaPor]);
             sendDataConst(pxParam, "\r\n\r\n\r\n");
             }
             else if(pxParam->resposta.bSemResposta)
             {
             logwifidiag(pxParam, "*** sem
resposta!!!  \r\n\r\n\r\n");
             }
             else if(pxParam->resposta.bAchouErro)
             {
             logwifidiag(pxParam, "*** achou
erro!!!  \r\n\r\n\r\n");
             }
             else
             {
             logwifidiag(pxParam, "*** ????????????
\r\n\r\n\r\n");
             }

          if ((!pxParam->resposta.bAchouTexto) || (pxParam-
>resposta.nIndiceSubStrEsperaPor == 3) ) //3 = "CLOSED"
          {
          vTaskDelay(1);
```

```
          //Se deu muitos erros já, reconectar até com a rede
          if( QtdErrosComunicacaoHost++ >
CLIENTEHTTP_QTD_ERROS_HOST_PARA_RECONECTAR_REDE)
          {
          sendDataConst(pxParam, "AT+CWQAP\r\n");
          conectado = 0;
          }
          continue;

          }

          EntraSC(); leds_pisca(LEDS_LED_LARANJA);  SaiSC();


             /////////////////////////////////////// Envio
da requisição com os dados ////////////////////////////////


          xQueueReset(pxParam->filas.RecepcaoDoModem.xCola);

          //checa tamanho
          if(!testaTextosCabemNoItemDaFilaTxModem(pxParam,
"AT+CIPSEND=\r\n", szTam, NULL))
          {
          sendDataConst(pxParam, "AT+CIPCLOSE\r\n");
          continue;
          }


          taskENTER_CRITICAL();

          sprintf(pxParam->filas.szBufferItem,
"AT+CIPSEND=%s\r\n", szTam);

          taskEXIT_CRITICAL();

          xQueueSendToBack(pxParam-
>filas.TranmissaoParaModem.xCola, (void *)pxParam-
>filas.szBufferItem, portMAX_DELAY);
             //deix de espear ERO e foca em esperar o prompt.
          if(esperaPor(pxParam, ">", "", "", timeout,
tentivasEsperaResp) != RECEBEU_TEXTO)
             {
          //debug apenas:
          vTaskDelay(1000 / portTICK_PERIOD_MS);

          logwifidiag(pxParam, "** Erro 1 ** Saindo pois não
recebeu o prompt para tx \">\" ***");
          sendDataConst(pxParam, "AT+CIPCLOSE\r\n");


          vTaskDelay(1);
```

```
          //Se deu muitos erros já, reconectar até com a rede
          if( QtdErrosComunicacaoHost++ >
CLIENTEHTTP_QTD_ERROS_HOST_PARA_RECONECTAR_REDE)
          conectado = 0;
          continue;

           }

           EntraSC(); leds_pisca(LEDS_LED_LARANJA);  SaiSC();


           sendData(pxParam, pxParam-
>requisicao.pcBufferRequisicao);
           if (esperaPor(pxParam, "SEND OK", "200 OK", "ERROR",
timeout,
          3 * tentivasEsperaResp) != RECEBEU_TEXTO)
           {
          //debug apenas:
          vTaskDelay(1000 / portTICK_PERIOD_MS);

          //debug apenas:
          logwifidiag(pxParam, "** Erro 2 ** Saindo pois não
recebeu ok da transmissão **** \r\n");

          vTaskDelay(1);

          //Se deu muitos erros já, reconectar até com a rede
          if( QtdErrosComunicacaoHost++ >
CLIENTEHTTP_QTD_ERROS_HOST_PARA_RECONECTAR_REDE)
          {
          sendDataConst(pxParam, "AT+CWQAP\r\n");
          conectado = 0;
          }
          continue;
           }

           EntraSC(); leds_pisca(LEDS_LED_LARANJA);  SaiSC();
           //tudo certo. Sai do loop de tentativas
          //debug apenas:
          logwifidiag(pxParam, "** OK ** Tudo certo. Sai do
loop de tentativas **** \r\n");

          //tira item enviado da fila
          xQueueReceive(pxParam->filas.DatosEnvioHttp.xCola,
          (void *)pxParam->filas.szBufferItem, 1000 /
portTICK_PERIOD_MS );

          taskENTER_CRITICAL();
          leds_pisca(LEDS_LED_AZUL);
          taskEXIT_CRITICAL();
```

```
          QtdErrosComunicacaoHost = 0;

          break;


          // sendDataConst("AT+CIPCLOSE\r\n"); //deixa conexão
com o host aberta

          } // while(testativasEnvio--)

          //SE NÃ ODEU, RETIRA ITEM MESM OCOM PROBLEMA
          if(!testativasEnvio)
          {
          //tira item enviado da fila
          xQueueReceive(pxParam->filas.DatosEnvioHttp.xCola,
          (void *)pxParam->filas.szBufferItem, 1000 /
portTICK_PERIOD_MS );


          taskENTER_CRITICAL();

          for(i=3; i; i--)
          {
          EntraSC();
          leds_pisca(LEDS_LED_AZUL);
          SaiSC();
          vTaskDelay(333 / portTICK_PERIOD_MS);
          }

          taskEXIT_CRITICAL();

          }

          }

          }
```

task_cliente_http.h

```
/*
 * task_cliente_http.h
 *
 *  Created on: 6 de mai de 2018
 *      Author: Max
 */

#ifndef APLICACAO_TASKS_TASK_CLIENTE_HTTP_H_
#define APLICACAO_TASKS_TASK_CLIENTE_HTTP_H_


#ifdef ARDUINO
#include <Arduino_FreeRTOS.h>
#else
#include <FreeRTOS.h>
#endif

#include <queue.h>
#include <task.h>
#include "global.h"


#define TASKCLIHTTP_TAM_BUFFER_FILTROS 17

/*
 * @brief Cria a task
 * @param QueueHandle_t xColaDadosEnvioHttp - Estes são os dados a serem enviados
 * @param size_t nTamMaxBufferDadosEnvioHttp - tamanho do buffer com o item da fila a ser enviado
 * @param QueueHandle_t xColaRecepcaoLinha - Fila de onde vem os textos a nível de linha
 * @param QueueHandle_t xColaEnvioLinha - fila para enviar texto (da task_serial, por exemplo)
 * @param size_t nTamMaxBufferRecepcao - Tamamnho do buffer de recepçao
 * @param size_t nTamMaxBufferEnvio - Indica o limite de buffer (usado apra garantir nulo no final
 * @param char *szNomeRedeWifi
 * @param char *szSenhaRedeWifi
 * @param char *szIPDestino
 * @param char *szPortaDestino
 * @param int nQtdFiltrosEliminacaoFilaEntrada - indica qunatos filtros de substring existem apra eliminar (ou seja,
 * deixar de pegar o item da fila de entrada e esperar que outra task pegue),
```

```
          * @param char *pszFiltrosEliminacaoFilaEntrada -
Ponteiro apra buffer com TASKCLIHTTP_TAM_BUFFER_FILTROS
elementos,
          * sendo o caracter nulo o divbisor de um filtro para
outro. Os filtros serão procurados como substring no
          * valor obtido da fila, se achar não enviará e
alguma outra task precisa retirá-lo.
          */
          #ifndef _TASKCLIHTTP_C_
          extern
          #endif
          TaskHandle_t taskclientehttp_create(QueueHandle_t
xColaDadosEnvioHttp,
          size_t nTamMaxBufferDadosEnvioHttp,
          QueueHandle_t xColaRecepcaoLinha, QueueHandle_t
xColaEnvioLinha,
          size_t nTamMaxBufferRecepcao, size_t
nTamMaxBufferEnvio, char *szNomeRedeWifi,
          char *szSenhaRedeWifi, char *szIPDestino, char
*szPortaDestino,
          int nQtdFiltrosEliminacaoFilaEntrada, char
*pszFiltrosEliminacaoFilaEntrada);


          #endif /* APLICACAO_TASKS_TASK_CLIENTE_HTTP_H_ */
```

task_ecoserial.c

```
/*
 * task_ecoserial.c
 *
 *  Created on: 20 de abr de 2018
 *      Author: Max
 */

#define _TASKECOSERIAL_C_

#include "task_ecoserial.h"
#include <string.h>
#include "util_memory.h"

static TaskHandle_t gxTaskEchoHandle = NULL;


static volatile QueueHandle_t
gxColaRecepcaoTaskEcoSerial = NULL;
static volatile QueueHandle_t
TaskEcoSerialTaskEcoSerial = NULL;
static volatile size_t gnTamMaxBufferRecepcao = 0;
static volatile size_t gnTamMaxBufferEnvio = 0;

static char *gszBufferRxTaskEcoSerial = NULL;
static char *gszBufferTx = NULL;


/*
 * @brief função de loop da task de eco da serial
 * @param pvParameters
 * @retval TaskHandle_t
 */
void taskEcoSerialFunc( void *pvParameters );


/*
 * @brief Cria a task
 * @param QueueHandle_t xColaRecepcaoLinha - Fila de
onde vem os textos a nível de linha
 * @param QueueHandle_t xColaEnvioLinha - fila para
enviar texto (da task_serial, por exemplo)
 * @param size_t nTamMaxBufferRecepcao - Tamamnho do
buffe de recepçao
 * @param size_t nTamMaxBufferEnvio - Indica o limite
de buffer (usado apra garantir nulo no final
 * de textos grandes demais)
 */
TaskHandle_t taskecoserial_create(QueueHandle_t
xColaRecepcaoLinha, QueueHandle_t xColaEnvioLinha,
```

```
          size_t nTamMaxBufferRecepcao, size_t
nTamMaxBufferEnvio)
          {

          gxColaRecepcaoTaskEcoSerial = xColaRecepcaoLinha;
          TaskEcoSerialTaskEcoSerial = xColaEnvioLinha;
          gnTamMaxBufferEnvio = nTamMaxBufferEnvio;
          gnTamMaxBufferRecepcao = nTamMaxBufferRecepcao;

          gszBufferRxTaskEcoSerial =
pvPortMalloc(gnTamMaxBufferRecepcao);
          gszBufferTx = pvPortMalloc(gnTamMaxBufferEnvio);

          xTaskCreate(taskEcoSerialFunc,
                  TASKECOSERIAL_NOME,
          TASKECOSERIAL_STACK_SIZE,
          NULL,
          TASKECOSERIAL_PRIORITY,
          &gxTaskEchoHandle );

          return gxTaskEchoHandle;

          }

          /*
           * @brief Código da task propriamente dito
           * @param void *pvParameters - Não uso
           * @retval None
           */
          void taskEcoSerialFunc( void *pvParameters )
          {
          const char *szMsgDesligamento = "{\"msg\": \"Echo
desativado!\" }\r\n";
          for (;;)
          {

          //pvParameters
          if(gxColaRecepcaoTaskEcoSerial &&
          (xQueueReceive(gxColaRecepcaoTaskEcoSerial, (void
*)gszBufferRxTaskEcoSerial, portMAX_DELAY ) == pdPASS))
          {
          for(int i=0; i< (gnTamMaxBufferRecepcao-7); i++)
          {
          if(gszBufferRxTaskEcoSerial[i+0] == 'e' &&
            gszBufferRxTaskEcoSerial[i+1] == 'c' &&
            gszBufferRxTaskEcoSerial[i+2] == 'h' &&
            gszBufferRxTaskEcoSerial[i+3] == 'o' &&
            gszBufferRxTaskEcoSerial[i+4] == 'o' &&
            gszBufferRxTaskEcoSerial[i+5] == 'f' &&
            gszBufferRxTaskEcoSerial[i+6] == 'f')
```

```
          //if(strcmp(szBufferRx, "echooff") == 0)
          {

          /* xQueueSendToBack(TaskEcoSerialTaskEcoSerial,
          (void *)
          utilmem_copiarTextoConst(gszBufferTx,
          msgDesligamento, gnTamMaxBufferEnvio),
          portMAX_DELAY);
          */
          xQueueSendToBack(TaskEcoSerialTaskEcoSerial, (void *)
szMsgDesligamento, portMAX_DELAY);


          //max: aqui tinha um problema de pilha devido a esta
chamada a utilmem_xxx
          //aumentei bastante mas retoranva 0 em pilhaSobrando,
agora comentando
          //a chamada utilmem_copiarTextoConst sobrou 164.
(defini INCLUDE_uxTaskGetStackHighWaterMark 1 em
          //FreeRTOSConfig.h para habilitar isto:
          // ficou com menos pilha e sem a chamada..
          // UBaseType_t pilhaSobrando =
uxTaskGetStackHighWaterMark(gxTaskEchoHandle);
          //
          // pilhaSobrando++;
          // pilhaSobrando--;

          vTaskSuspend(NULL);

          break;
          }
          }

          if(TaskEcoSerialTaskEcoSerial)
          {

          xQueueSendToBack(TaskEcoSerialTaskEcoSerial,
          (void *) utilmem_copiarTextoConst(gszBufferTx, "
{\"echo\": \"", gnTamMaxBufferEnvio), portMAX_DELAY);

          if(gnTamMaxBufferEnvio < gnTamMaxBufferRecepcao)
          gszBufferRxTaskEcoSerial[gnTamMaxBufferEnvio-1] = 0;

          xQueueSendToBack(TaskEcoSerialTaskEcoSerial, (void
*)gszBufferRxTaskEcoSerial, portMAX_DELAY);

          xQueueSendToBack(TaskEcoSerialTaskEcoSerial,
          (void *) utilmem_copiarTextoConst(gszBufferTx,
"\"}\r\n", gnTamMaxBufferEnvio), portMAX_DELAY);

          }
          }
```

```
  }
}
```

task_ecoserial.h

```
/*
 * task_ecoserial.h
 *
 *  Created on: 20 de abr de 2018
 *      Author: Max
 */

#ifndef APLICACAO_TASKS_TASK_ECOSERIAL_H_
#define APLICACAO_TASKS_TASK_ECOSERIAL_H_

#ifdef ARDUINO
#include <Arduino_FreeRTOS.h>
#else
#include <FreeRTOS.h>
#endif

#include <queue.h>
#include <task.h>
#include "global.h"



/*
 * @brief Cria a task
 * @param QueueHandle_t xColaRecepcaoLinha - Fila de
onde vem os textos a nível de linha
 * @param QueueHandle_t xColaEnvioLinha - fila para
enviar texto (da task_serial, por exemplo)
 * @param size_t nTamMaxBufferRecepcao - Tamamnho do
buffe de recepçao
 * @param size_t nTamMaxBufferEnvio - Indica o limite
de buffer (usado apra garantir nulo no final
 * de textos grandes demais)
 */
#ifndef _TASKECOSERIAL_C_
extern
#endif
TaskHandle_t taskecoserial_create(QueueHandle_t
xColaRecepcaoLinha, QueueHandle_t xColaEnvioLinha,
size_t nTamMaxBufferRecepcao, size_t
nTamMaxBufferEnvio);

#endif /* APLICACAO_TASKS_TASK_ECOSERIAL_H_ */
```

task_giroscopio.c

```
/*
 * task_giroscopio.c
 *
 *  Created on: 5 de maio de 2018
 *      Author: Max
 */

#define _TASKGIRO_C_

#include "task_giroscopio.h"
#include <stdio.h>
#include <string.h>
#ifdef ARDUINO
#include "tipos.h"
#include "giroscopio.h"
#else
#include "plataforma/tipos.h"
#include <plataforma/includes/giroscopio.h>
#endif
#include "util_memory.h"
#include "global.h"

#include <math.h>

#define ENVIAR_LOG 0

#if(ENVIAR_LOG == 0)
#define log(S)
#define logval(S, I)
#define logvals(S, I)
#define logdados(S, I)
#else
#ifdef ARDUINO
extern void printlog(int qtdParams, const char *s,
int valor);
extern void printlogss(const char *s, char * valor);

#define log(S) printlog(1, S, 0)
#define logval(S, I) printlog(2, S, I)
#define logvals(S, I) printlogss(S, I)
#define logdados(S, I) printlog(S, I)
#else
#define log(S)
#define logval(S, I)
#define logvals(S, I)
#define logdados(S, I)
#endif
#endif
```

```
          #define ABS(x)                    ( (x < 0) ? (-x) :
x )

          typedef struct
          {
          giroscopio_leitura_t xleitura;
          const char *szMascaraPacotes;
 //[GLOBAL_MAX_MASCARA_GIROSCOPIO];
          taskgiroscopio_paraminit_t xParamInit;
          QueueHandle_t xColaGiroscopio;

          } param_funcaotask_giroscopio_t;

          typedef union {
          struct {
          uint32_t nTipoParametro         : 4; //defines
TIPOPARAMTASKNOTIFGIRO_...
          uint32_t nValor8bits            : 8; //parametros de
8 bits de acordo com tipo
          uint32_t nValor20bits          : 20; //parametros de
8 bits de acordo com tipo
          };
          uint32_t nValor;
          } param_notifocacaotask_giroscopio_t;


          TaskHandle_t gxTaskGiroscopioHandle;

          static void taskGiroscopioFunc( void *pvParameters
__attribute__((unused)) );



          /*
           * @brief Cria a task
           * @param taskgiroscopio_paraminit_t *pxParam -
Descreve os parametros de funcionamento.
           * Os dados passados por ponteiro serão copiados no
módulo.
           * @retval TaskHandle_t - Task de tratamento dos
dados criada
           */
          TaskHandle_t
taskgiroscopio_create(taskgiroscopio_paraminit_t *pxParam)
          {

             log("taskgiroscopio_create() - inicio");
             logval("heap size atual: ",
xPortGetFreeHeapSize());  log("criando param task");
```

```
            param_funcaotask_giroscopio_t *pxParamTask =
pvPortMalloc( sizeof(param_funcaotask_giroscopio_t) );

            memcpy(&(pxParamTask->xParamInit), pxParam,
sizeof(taskgiroscopio_paraminit_t));

            //pxParamTask->szMascaraPacotes =
pvPortMalloc(GLOBAL_MAX_MASCARA_GIROSCOPIO * sizeof(char));
            //
            //strcpy(pxParamTask->szMascaraPacotes, pxParam-
>szMascaraPacotes);
            //pxParam->szMascaraPacotes = NULL; //deixa de usar
esta ref. que será invalida ao sair do escopo

            //só copia a referencia
            pxParamTask->szMascaraPacotes = pxParam-
>szMascaraPacotes;

             logval("heap size atual: ",
xPortGetFreeHeapSize());  log("criando task AppGiroscopio");
          xTaskCreate(taskGiroscopioFunc,
                TASKGIROSCOPIO_NOME,
          TASKGIROSCOPIO_STACK_SIZE,
          (void *) pxParamTask,
          TASKGIROSCOPIO_PRIORITY,
          &gxTaskGiroscopioHandle );


          //inicia o hardware
             logval("heap size atual: ",
xPortGetFreeHeapSize());  log("init. hw");
          giroscopio_init(pxParam->xTipoFiltro == gtfSemFiltro
? 0.0 : pxParam->fValorMinimoLeituras);

          log("taskgiroscopio_create() - fim");
             return gxTaskGiroscopioHandle;
          }


          /*
           * @brief Permite reconfigurar a task após iniciada,
para mudar por exemplo o filtro atual de leituras
           * @param *taskgiroscopio_paraminit_t pxParam -
Descreve os parametros de funcionamento
           * @param uint8_t nTipoAlteracao - Indica o que deve
ser atualizado (suporta apenas os valores dos defines
           * @retval int - retorna verdadeiro se conseguiu
enviar o comand ode reconfig. para a task
TIPOPARAMTASKNOTIFGIRO_...)
           */
```

```
          #ifndef _TASKGIRO_C_
          extern
          #endif
          int
taskgiroscopio_reconfigurar(taskgiroscopio_paraminit_t
*pxParam, uint8_t nTipoAlteracao)
          {
          if(!gxTaskGiroscopioHandle)
          return 0;

          //suporto por hora a mudança de filtro por hora, que
é oque me interessa
          param_notifocacaotask_giroscopio_t
xParamNotificacaoTask;

          xParamNotificacaoTask.nTipoParametro =
nTipoAlteracao;
          switch(xParamNotificacaoTask.nTipoParametro)
          {
             case TIPOPARAMTASKNOTIFGIRO_TIPOFILTRO:
          xParamNotificacaoTask.nValor8bits    = (uint8_t)
pxParam->xTipoFiltro;
          xParamNotificacaoTask.nValor20bits   = pxParam-
>fValorMinimoLeituras;
          break;

             case TIPOPARAMTASKNOTIFGIRO_TEMPOESPERALEITURAS:
          xParamNotificacaoTask.nValor20bits    = pxParam-
>xTempoEsperaEntreLeituras;
          break;

             default:
          return 0;
          }

          return xTaskNotify(gxTaskGiroscopioHandle,
xParamNotificacaoTask.nValor, eSetValueWithOverwrite);
          }

          /*
           * monta pacote com a leitura
           */
          #ifdef VERSAO_PONTO_FLUTUANTE
          void montarPacoteLeitura(char *szPacoteDestino, const
char *szMascara, giroscopio_leitura_t *pxLeitura)
          {
          double *leitura[3];
          double dif;
          unsigned int nEixoInt[3];
          unsigned int nEixoFrac[3];
```

```
          char sinal[3];
          int i;
          char szValores[3][20];

          leitura[0] = &pxLeitura->nEixoX;
          leitura[1] = &pxLeitura->nEixoY;
          leitura[2] = &pxLeitura->nEixoZ;

          for(i=0; i<3; i++)
          {
          sinal[i] = ' ';
          if( (*leitura[i]) < 0 )
          {
          *leitura[i] *= (float)-1.0;
          sinal[i] = '-';
          }

          // printf("-- leitura -> %f -- ", *leitura[i]);
          nEixoInt[i]  = (unsigned int)trunc( *leitura[i] );
          dif = *leitura[i] - nEixoInt[i];
          nEixoFrac[i] = (unsigned int)trunc(  dif * 10000 );


          sprintf(szValores[i], "%c%d.%d",
 sinal[i],nEixoInt[i], nEixoFrac[i]);
          /*
          sprintf(szValores[i], "
[%c|%u.|%u|%u|%u|%u]",  sinal[i],nEixoInt[i],
            (nEixoFrac[i]/1000)%10,
            (nEixoFrac[i]/100)%10,
            (nEixoFrac[i]/10)%10,
            (nEixoFrac[i]/1)%10
          );
          */

          }

          sprintf(szPacoteDestino, szMascara, szValores[0],
szValores[1], szValores[2],
          pxLeitura->xTempoEntreLeituras * portTICK_PERIOD_MS);

          }
          #endif

          void montarPacoteLeitura(char *szPacoteDestino, const
char *szMascara, giroscopio_leitura_t *pxLeitura)
          {
          unsigned int nEixoInt;
          char szValores[3][6];

          for(int i=0; i<3; i++)
```

```
          {
          switch(i)
          {
          case 0: nEixoInt  = (unsigned int)trunc( pxLeitura-
>nEixoX ); break;
          case 1: nEixoInt  = (unsigned int)trunc( pxLeitura-
>nEixoY ); break;
          case 2: nEixoInt  = (unsigned int)trunc( pxLeitura-
>nEixoZ ); break;
          }
          //limita a 5 digitos
          nEixoInt  = nEixoInt% 100000;
          sprintf(szValores[i], "%d",  nEixoInt );
          }

          sprintf(szPacoteDestino, szMascara, szValores[0],
szValores[1], szValores[2],
          pxLeitura->xTempoEntreLeituras * portTICK_PERIOD_MS);

          }


          /*
           * @brief Código da task propriamente dito (para
tratar o giroscopio)
           * @param void *pvParameters - Não uso
           * @retval None
           */
          void taskGiroscopioFunc( void *pvParameters )
          {

          param_notifocacaotask_giroscopio_t xParamNotificacao;
          param_funcaotask_giroscopio_t *pxParamTask =
(param_funcaotask_giroscopio_t *) pvParameters;
           //TickType_t xTempoEntreAcumulado = 0;
          char szPacote[GLOBAL_TAMANHO_PACOTE_GIROSCOPIO];
          int bLeituraAceita;

          //log("taskGiroscopioFunc - iniciou!");

          for (;;)
          {

          if(xTaskNotifyWait(
          0, //ulBitsToClearOnEntry
          0, //ulBitsToClearOnExit
          &xParamNotificacao.nValor, //pulNotificationValue
          0) == pdPASS)
          {
```

```
          logval("Notificacao task giro - tipo: ",
xParamNotificacao.nTipoParametro);

          //trata o tipo de parâmetro
          switch(xParamNotificacao.nTipoParametro)
          {
            case  TIPOPARAMTASKNOTIFGIRO_NENHUM:
            //nada
            break;

            case  TIPOPARAMTASKNOTIFGIRO_TIPOFILTRO:
          pxParamTask->xParamInit.xTipoFiltro =
xParamNotificacao.nValor8bits;
          pxParamTask->xParamInit.fValorMinimoLeituras =
(float) xParamNotificacao.nValor20bits;

          logval("Tipo de filtro alterado para: ", pxParamTask-
>xParamInit.xTipoFiltro);
            break;

            case  TIPOPARAMTASKNOTIFGIRO_TEMPOESPERALEITURAS:
          pxParamTask->xParamInit.xTempoEsperaEntreLeituras =
xParamNotificacao.nValor20bits;

          logval("xTempoEsperaEntreLeituras alterado para: ",
pxParamTask->xParamInit.xTempoEsperaEntreLeituras);
             break;

          }
          }

          //logval("Espera (ms): ", pxParamTask-
>xParamInit.xTempoEsperaEntreLeituras * portTICK_PERIOD_MS);
          //espera entre leituras
          vTaskDelay(pxParamTask-
>xParamInit.xTempoEsperaEntreLeituras);

          //log("Depois da espera...");

            //testa ler um valor do giroscopio
            if( giroscopio_lerUltimoValor(&(pxParamTask-
>xleitura)) )
            {
            bLeituraAceita = 1;
            switch(pxParamTask->xParamInit.xTipoFiltro)
            {
            case gtfSemFiltro:
              log("[Retornou leitura - gtfSemFiltro]");
              break;

            case gtfMinimoValorAlgumEixo:
```

```
              log("[Retornou leitura -
gtfMinimoValorAlgumEixo]");
                 //já filtrado pelo driver
              break;

             case gtfEliminaAbaixoDoMinimoValor:
             {
              log("[Retornou leitura -
gtfEliminaAbaixoDoMinimoValor]");
              int nEixosAcima = 3;
           if( ABS(pxParamTask->xleitura.nEixoX) < pxParamTask-
>xParamInit.fValorMinimoLeituras )
           {
            log("** Leitura em X descartada");
            pxParamTask->xleitura.nEixoX = 0;
            nEixosAcima--;
           }
           if( ABS(pxParamTask->xleitura.nEixoY) < pxParamTask-
>xParamInit.fValorMinimoLeituras )
           {
            log("* Leitura em Y descartada");
            pxParamTask->xleitura.nEixoY = 0;
            nEixosAcima--;
           }
           if( ABS(pxParamTask->xleitura.nEixoZ) < pxParamTask-
>xParamInit.fValorMinimoLeituras )
           {
            log("* Leitura em Z descartada");
            pxParamTask->xleitura.nEixoZ = 0;
            nEixosAcima--;
           }

              if( nEixosAcima == 0 )
              {
               log("* Toda a leitura foi descartada");
                  bLeituraAceita = 0;
              }
             }
              break;

             case gtfMinimoValorTodosEixo:
              log("[Retornou leitura -
gtfMinimoValorTodosEixo]");
              if( (ABS(pxParamTask->xleitura.nEixoX) <
pxParamTask->xParamInit.fValorMinimoLeituras) ||
              (ABS(pxParamTask->xleitura.nEixoY) < pxParamTask-
>xParamInit.fValorMinimoLeituras) ||
           (ABS(pxParamTask->xleitura.nEixoZ) < pxParamTask-
>xParamInit.fValorMinimoLeituras) )
```

```
            {
              log("** Leitura descartada");
                 bLeituraAceita = 0;
            }
            break;

          case gtfMinimoValorEixoX:
            log("[Retornou leitura - gtfMinimoValorEixoX]");
            if( ABS(pxParamTask->xleitura.nEixoX) <
pxParamTask->xParamInit.fValorMinimoLeituras )
            {
              log("** Leitura descartada");
                 bLeituraAceita = 0;
            }
            break;

          case gtfMinimoValorEixoY:
            log("[Retornou leitura - gtfMinimoValorEixoY]");
            if( ABS(pxParamTask->xleitura.nEixoY) <
pxParamTask->xParamInit.fValorMinimoLeituras )
            {
              log("** Leitura descartada");
                 bLeituraAceita = 0;
            }
            break;

          case gtfMinimoValorEixoZ:
            log("[Retornou leitura - gtfMinimoValorEixoZ]");
            if( ABS(pxParamTask->xleitura.nEixoZ) <
pxParamTask->xParamInit.fValorMinimoLeituras )
            {
              log("** Leitura descartada");
                 bLeituraAceita = 0;
            }
            break;

          case gtfApenasValorEixoX:
            log("[Retornou leitura - gtfApenasValorEixoX]");
            if( ABS(pxParamTask->xleitura.nEixoX) <
pxParamTask->xParamInit.fValorMinimoLeituras )
            {
              log("** Leitura descartada");
                 bLeituraAceita = 0;
            }
            else
            {
            pxParamTask->xleitura.nEixoY = 0;
          pxParamTask->xleitura.nEixoZ = 0;
            }
```

```
            break;

           case gtfApenasValorEixoY:
            log("[Retornou leitura - gtfApenasValorEixoY]");
            if( ABS(pxParamTask->xleitura.nEixoY) <
pxParamTask->xParamInit.fValorMinimoLeituras )
            {
              log("** Leitura descartada");
                 bLeituraAceita = 0;
            }
            else
            {
            pxParamTask->xleitura.nEixoX = 0;
          pxParamTask->xleitura.nEixoZ = 0;
            }
            break;

           case gtfApenasValorEixoZ:
            log("[Retornou leitura - gtfApenasValorEixoZ]");
            if( ABS(pxParamTask->xleitura.nEixoZ) <
pxParamTask->xParamInit.fValorMinimoLeituras )
            {
              log("** Leitura descartada");
                 bLeituraAceita = 0;
            }
            else
            {
            pxParamTask->xleitura.nEixoX = 0;
          pxParamTask->xleitura.nEixoY = 0;
            }
            break;

           case gtfUsarFuncaoFiltro:
            log("[Retornou leitura - gtfUsarFuncaoFiltro]");
           if(pxParamTask->xParamInit.pCallbackFiltro)
           {
           if( !pxParamTask->xParamInit.pCallbackFiltro(&
(pxParamTask->xleitura)) )
           {
           log("** Leitura descartada");
           }
           }
           }

            if(!bLeituraAceita)
         continue;

            if(!pxParamTask->xParamInit.xColaPacotes)
         continue;
```

```
          /*
            if(pxParamTask->xParamInit.xTempoEntreLeituras &&
               (pxParamTask->xParamInit.xTempoEntreLeituras !=
pxParamTask->xParamInit.xTempoEsperaEntreLeituras) )
            {
            xTempoEntreAcumulado += pxParamTask-
>xleitura.xTempoEntreLeituras;
            if(xTempoEntreAcumulado < pxParamTask-
>xParamInit.xTempoEntreLeituras)
            continue;

            xTempoEntreAcumulado = 0;
            }

          */


          taskENTER_CRITICAL();
          montarPacoteLeitura(szPacote, pxParamTask-
>szMascaraPacotes, &pxParamTask->xleitura);
          taskEXIT_CRITICAL();

          xQueueSendToBack(pxParamTask-
>xParamInit.xColaPacotes, szPacote, 0);

              }
          else
          {
          log("[Não Retornou leitura]");
          }

            }
          }
```

task_giroscopio.h

```
/*
 * task_giroscopio.h
 *
 *  Created on: 5 de maio de 2018
 *      Author: Max
 */

#ifndef APLICACAO_TASKS_TASK_GIROSCOPIO_H_
#define APLICACAO_TASKS_TASK_GIROSCOPIO_H_

#include "global.h"
#include <queue.h>
#include <task.h>
#ifdef ARDUINO
#include "tipos.h"
#else
#include <tipos.h>
#endif

//função de callback para filtro
typedef BaseType_t (*callbackFiltroGiroscopio_t)
(giroscopio_leitura_t *xleitura);


typedef enum {
gtfSemFiltro,                   //Aceita todas as
leituras
gtfMinimoValorAlgumEixo,        //Aceita a leitura
se houver valor mínimo em algum eixo
gtfEliminaAbaixoDoMinimoValor,  //Aceita a leitura
se houver valor mínimo em algum eixo e zera os abaixo
   gtfMinimoValorTodosEixo,        //Aceita a
leitura se houver valor mínimo em TODOS os eixos

gtfMinimoValorEixoX,            //Aceita a leitura
se houver valor mínimo no eixo X, mantendo as demais com seus
valores
gtfApenasValorEixoX,            //Aceita a leitura
se houver valor mínimo no eixo X, e descarta as demais leituras

gtfMinimoValorEixoY,            //Aceita a leitura
se houver valor mínimo no eixo Y, mantendo as demais com seus
valores
gtfApenasValorEixoY,            //Aceita a leitura
se houver valor mínimo no eixo Y, e descarta as demais leituras

gtfMinimoValorEixoZ,            //Aceita a leitura
se houver valor mínimo no eixo Z, mantendo as demais com seus
```

```
valores
          gtfApenasValorEixoZ,             //Aceita a leitura
se houver valor mínimo no eixo Z, e descarta as demais leituras

          gtfUsarFuncaoFiltro              //chama funcao de
callback que pdoe alterar leitura e se ela retornar verdadeiro
aceita a leitura
                                  //assim delega o filtro apra
um nível mais acima da aplicação

          } taskgiroscopio_tipofiltro_t;

          typedef struct {
          const char *szMascaraPacotes; //mascara para gerar o
pacotes na fila (def. no param. sequinte).
          //deve ter até GLOBAM_MAX_MASCARA_GIROSCOPIO
caracteres, incluindo o nulo
          QueueHandle_t xColaPacotes; //fila onde os pacotes
gerados com base no texto de formatação são colocados
          //deve aceitar array de caractees com o tamanho
GLOBAL_TAMANHO_PACOTE_GIROSCOPIO
          TickType_t xTempoEntreLeituras; //para filtrar
leituras. Se 0 todas são aceitas, senão apenas quando o
intervalo
          //acumulado fo maior ou igual a este valor será pego
o valor. ***> tempo em tick timer do SO. Usar
          //3000 / portTICK_PERIOD_MS para 3 segundos, por
exemplo.
          TickType_t xTempoEsperaEntreLeituras; //ja este
indica dw quanto em quanto tempo vai tentar ler algo
          taskgiroscopio_tipofiltro_t xTipoFiltro; //indica se
vai filtrar e como
          float fValorMinimoLeituras; //valor minimo em rad/s
para aceitar a leitura em um eixo (ver
taskgiroscopio_tipofiltro_t xTipoFiltro)
          callbackFiltroGiroscopio_t pCallbackFiltro; //indica
a função a ser chaamda para filtro (ver
taskgiroscopio_tipofiltro_t xTipoFiltro)

          } taskgiroscopio_paraminit_t;



          /*
           * @brief Cria a task
           * @param *taskgiroscopio_paraminit_t pxParam -
Descreve os parametros de funcionamento
           * @retval TaskHandle_t - Task de tratamento dos
dados criada
```

```
          */
          #ifndef _TASKGIRO_C_
          extern
          #endif
          TaskHandle_t
taskgiroscopio_create(taskgiroscopio_paraminit_t *pxParam);

          /*
           * Tipos de alterações
           */
          #define TIPOPARAMTASKNOTIFGIRO_NENHUM 0
          #define TIPOPARAMTASKNOTIFGIRO_TIPOFILTRO 1
          #define TIPOPARAMTASKNOTIFGIRO_TEMPOESPERALEITURAS 2

          /*
           * @brief Permite reconfigurar a task após iniciada,
para mudar por exemplo o filtro atual de leituras
           * @param *taskgiroscopio_paraminit_t pxParam -
Descreve os parametros de funcionamento
           * @param uint8_t nTipoAlteracao - Indica o que deve
ser atualizado (suporta apenas os valores dos defines
           * @retval int - retorna verdadeiro se conseguiu
enviar o comand ode reconfig. para a task
TIPOPARAMTASKNOTIFGIRO_...)
           */
          #ifndef _TASKGIRO_C_
          extern
          #endif
          int
taskgiroscopio_reconfigurar(taskgiroscopio_paraminit_t
*pxParam, uint8_t nTipoAlteracao);

          #endif /* APLICACAO_TASKS_TASK_GIROSCOPIO_H_ */
```

task_joystick.c

```
/*
 * task_joystick.c
 *
 *  Created on: 3 de mai de 2018
 *      Author: maxback
 */

#define _TASKJOYSTICK_C_

#include <stdio.h>
#include <string.h>
#include "task_joystick.h"

#ifdef ARDUINO
#include "tipos.h"
#include "joystick.h"
#else
#include "plataforma/tipos.h"
#include <plataforma/includes/joystick.h>
#endif

#include "global.h"
#include "util_memory.h"

static char
gszMascaraPacotes[GLOBAL_MAX_MASCARA_JOYSTICK];
static taskjoystick_paraminit_t gxParam;
static volatile QueueHandle_t gxColaJoystick = NULL;

static void taskJoystickFunc( void *pvParameters
__attribute__((unused)) );


/*
 * @brief Cria a task
 * @param taskjoystick_paraminit_t *pxParam -
Descreve os parametros de funcionamento.
 * Os dados passados por ponteiro serão copiados no
módulo.
 * @retval TaskHandle_t - Task de tratamento dos
dados criada
 */
TaskHandle_t
taskjoystick_create(taskjoystick_paraminit_t *pxParam)
{
  TaskHandle_t xTaskHandle;

   memcpy(&gxParam, pxParam,
sizeof(taskjoystick_paraminit_t));
```

```
            //gxParam.szMascaraPacotes =
utilmem_copiarTextoMenorTamanho(gszMascaraPacotes, pxParam-
>szMascaraPacotes,
            // sizeof(gszMascaraPacotes));
             // ** ficou com o local const char * horiginal
apontado

          xTaskCreate(taskJoystickFunc,
                TASKJOYSTICK_NOME,
          TASKJOYSTICK_STACK_SIZE,
          NULL,
          TASKJOYSTICK_PRIORITY,
          &xTaskHandle );

          gxColaJoystick = xQueueCreate(10,
sizeof(joystick_leitura_t));
          vQueueAddToRegistry( gxColaJoystick, "AppJoystick");

          //inicia o hardware
          joystick_init(gxColaJoystick);

          return xTaskHandle;

          }


          /*
           * @brief Código da task propriamente dito (para
tratar o acelerometro)
           * @param void *pvParameters - Não uso
           * @retval None
           */
          void taskJoystickFunc( void *pvParameters
__attribute__((unused)) )
          {
          TickType_t xTempoEntreAcumulado = 0;
          joystick_leitura_t leitura;
          char szPacote[GLOBAL_TAMANHO_PACOTE_JOYSTICK];

          for (;;)
          {
            //espera com a task parada até chegar algo na fila
            if(xQueueReceive( gxColaJoystick, (void *)&leitura,
portMAX_DELAY) == pdPASS)
            {
            if(gxParam.xTempoEntreLeituras)
            {
            xTempoEntreAcumulado +=
leitura.xTempoEntreLeituras;
```

```
            if(xTempoEntreAcumulado <
gxParam.xTempoEntreLeituras)
            continue;

            xTempoEntreAcumulado = 0;
            }

          if(!gxParam.xColaPacotes)
          continue;

          sprintf(szPacote, gszMascaraPacotes,
leitura.nLeituraX,
          leitura.nLeituraY);
              }
           }
          }
```

task_joystick.h

```
/*
 * task_joystick.h
 *
 *  Created on: 3 de mai de 2018
 *      Author: maxback
 */

#ifndef APLICACAO_TASKS_TASK_JOYSTICK_H_
#define APLICACAO_TASKS_TASK_JOYSTICK_H_

#ifdef ARDUINO
#include <Arduino_FreeRTOS.h>
#else
#include <FreeRTOS.h>
#endif

#include <queue.h>
#include <task.h>


typedef struct {
const char *szMascaraPacotes; //mascara para gerar o
pacotes na fila (def. no param. sequinte).
//deve ter até GLOBAM_MAX_MASCARA_JOYSTICK
caracteres, incluindo o nulo
QueueHandle_t xColaPacotes; //fila onde os pacotes
gerados com base no texto de formatação são colocados
//deve aceitar array de caractees com o tamanho
GLOBAL_TAMANHO_PACOTE_JOYSTICK
TickType_t xTempoEntreLeituras; //para filtrar
leituras. Se 0 todas são aceitas, senão apenas quando o
intervalo
//acumulado fo maior ou igual a este valor será pego
o valor. ***> tempo em tick timer do SO. Usar
//3000 / portTICK_PERIOD_MS para 3 segundos, por
exemplo.

} taskjoystick_paraminit_t;

/*
 * @brief Cria a task
 * @param *taskjoystick_paraminit_t pxParam -
Descreve os parametros de funcionamento
 * @retval TaskHandle_t - Task de tratamento dos
dados criada
 */
#ifndef _TASKJOYSTICK_C_
extern
```

```
          #endif
          TaskHandle_t
taskjoystick_create(taskjoystick_paraminit_t *pxParam);

          #endif /* APLICACAO_TASKS_TASK_JOYSTICK_H_ */
```

task_saudacao.c

```
/*
 * task_saudacao.c
 *
 *  Created on: 18 de abr de 2018
 *      Author: Max
 */

#define _TASKSAUDACAO_C_

#include "task_saudacao.h"
#include "task_serial.h"
#include "global.h"
#include "util_memory.h"

static TaskHandle_t gxTaskHandle = NULL;
//callback para envio
static volatile callbackEnvioTexto_t gpCallBackEnvio
= NULL;
static volatile size_t gnTamBufferLinha = 0;

/*
 * @brief função de loop da task serial
 * @param pvParameters
 * @retval TaskHandle_t
 */
void taskSaudacaoFunc( void *pvParameters );


/*
 * @brief Cria a task
 * @param QueueHandle_t xColaRecepcaoLinha - Fila de
onde vem os textos a nível de linha
 * @param  size_t nTamBufferLinha - Permite saber o
máximo de tamanho do buffe de string aceito.
 * @retval TaskHandle_t
 */
TaskHandle_t tasksaudacao_create(callbackEnvioTexto_t
callbackEnvio, size_t nTamBufferLinha){

gpCallBackEnvio = callbackEnvio;
gnTamBufferLinha = nTamBufferLinha;

if(!gpCallBackEnvio)
return NULL;

xTaskCreate(taskSaudacaoFunc,
      TASKSAUDACAO_NOME,
TASKSAUDACAO_STACK_SIZE,
NULL,
```

```
            TASKSAUDACAO_PRIORITY,
            &gxTaskHandle );

            return gxTaskHandle;

            }


            /*
             * @brief Código da task propriamente dito
             * @param void *pvParameters - Não uso
             * @retval None
             */
            void taskSaudacaoFunc( void *pvParameters
__attribute__((unused)) )
            {

            volatile char contador = '0';
               char szBuf[2];

            gpCallBackEnvio(
            "{\"msg\": \"STM32F4 Werable Controller
iniciado.\"}\r\n");

            gpCallBackEnvio("{\"msg\": \"Teste de saudacao ativo
e enviado a cada 1 seg. aprox.\"}\r\n");

            for (;;)
            {
            szBuf[0] = contador;
            szBuf[1] = '\0';

            gpCallBackEnvio("{\"msg\": \"Teste : ");
            gpCallBackEnvio(szBuf);
            gpCallBackEnvio("\"}\r\n");

               contador = (contador < 127) ? contador+1 : '0';

             vTaskDelay(1000 / portTICK_PERIOD_MS);
             }
            }
```

task_saudacao.h

```
/*
 * saudacao.h
 *
 *  Created on: 18 de abr de 2018
 *      Author: Max
 */

#ifndef APLICACAO_TASKS_TASK_SAUDACAO_H_
#define APLICACAO_TASKS_TASK_SAUDACAO_H_

#ifdef ARDUINO
#include <Arduino_FreeRTOS.h>
#else
#include <FreeRTOS.h>
#endif

#include <queue.h>
#include <task.h>
#include "global.h"



/*
 * @brief Cria a task
 * @param QueueHandle_t xColaRecepcaoLinha - Fila de
onde vem os textos a nível de linha
 * @param  size_t nTamBufferLinha - Permite saber o
máximo de tamanho do buffe de string aceito.
 * @retval TaskHandle_t
 */
#ifndef _TASKSAUDACAO_C_
extern
#endif
TaskHandle_t tasksaudacao_create(callbackEnvioTexto_t
callbackEnvio, size_t nTamBufferLinha);


#endif /* APLICACAO_TASKS_TASK_SAUDACAO_H_ */
```

task_serial.c

```
/*
 * task_serial.c
 *
 *  Created on: 18 de abr de 2018
 *      Author: Max
 */

#define _TASKSERIAL_C_
#include "task_serial.h"
#include <string.h>
#ifdef ARDUINO
#include "serial.h"
#include <Arduino.h>
#else
#include <plataforma/includes/serial.h>
#endif
#include <timers.h>
#include "util_memory.h"

#ifdef ARDUINO
extern void printlog(int qtdParams, const char *s,
int valor);
extern void printlogss(const char *s, char * valor);
#define log(S)
//printlog(1, S, 0);
#define logval(S, I)
//printlog(2, S, I);
#define logvals(S, I)
//printlogss(S, I);
#else
#define log(S)
#define logval(S, I)
#define logvals(S, I)
#endif

typedef struct {

QueueHandle_t xColaRecepcao;
QueueHandle_t xColaRecepcaoCaracteres;

//static volatile TimerHandle_t gxTimerRecepcao =
NULL;

char *szBufferRx;
size_t stTamanhoRx;
```

```
            size_t nTamItemFilaRecepcao;

            size_t nTamItemFilaRecepcaoCheia;
            callbackEnvioTexto_t CallbackFilaRecepcaoCheia;

            } param_dados_taskserial_rx_t;

            typedef struct {

            QueueHandle_t xColaEnvio;
            QueueHandle_t xColaEnvioCaracteres;

            char *szBufferTx;
            size_t nTamItemFilaEnvio;

            param_dados_taskserial_rx_t *paramRx;

            } param_dados_taskserial_tx_t;



            /*
             * @brief função de loop da task serial para Rx
             * @param pvParameters
             * @retval TaskHandle_t
             */
            void taskSerialRxFunc( void *pvParameters );


            /*
             * @brief função de loop da task serial para Tx
             * @param pvParameters
             * @retval TaskHandle_t
             */
            void taskSerialTxFunc( void *pvParameters );


            /*
             * @brief função de callback para o timeout entre
caracteres
             * @param pvParameters
             * @retval TaskHandle_t
             */
            void callbackTimeoutEntreCaracteresFunc(
TimerHandle_t xTimer );



            /*
             * @brief Cria a task
             * @param QueueHandle_t xColaEnvio - Fila de onde dem
os textos para envio
```

```
           * @param size_t nTamItemFilaEnvio - Tamanho do item
da fila de envio
           * @param size_t nQtdItensFilaEnvio - Quantos itens
tem a fila de envio
           * @param QueueHandle_t xColaRecepcao - Fila onde
deve colocar o texto recebido
           * @param size_t nTamItemFilaRecepcao - Tamanho do
item da fila de recepcao
           * @param size_t nQtdItensFilaRecepcao - Quantos
itens tem a fila de recepcao
           * @param uint32_t nTimeoutEntreCaracteresMs - tempo
em milesegundos entre caracteres
           * e Rx. Se pasa deste tempo termina a linha e envia
pela fila informada em xColaRecepcao
           * @param callbackEnvioTexto_t
callbackFilaRecepcaoCheia
           * @param size_t nTamItemFilaRecepcaoCheia - Tamanho
do item da fila de envio
           * @retval TaskHandle_t - Task de envio de dados
           */
          TaskHandle_t taskserial_create(
          QueueHandle_t xColaEnvio,
          size_t nTamItemFilaEnvio,
          size_t nQtdItensFilaEnvio,

          QueueHandle_t xColaRecepcao,
          size_t nTamItemFilaRecepcao,
          size_t nQtdItensFilaRecepcao,

          uint32_t nTimeoutEntreCaracteresMs,
callbackEnvioTexto_t callbackFilaRecepcaoCheia,
          size_t nTamItemFilaRecepcaoCheia)
          {
           TaskHandle_t xTaskHandle;

          //log("taskserial_create() - iniciando... ");


           param_dados_taskserial_rx_t *pxParamRx;
           param_dados_taskserial_tx_t *pxParamTx;

           pxParamRx = pvPortMalloc(
sizeof(param_dados_taskserial_rx_t) );
           pxParamTx = pvPortMalloc(
sizeof(param_dados_taskserial_tx_t) );


           pxParamRx->nTamItemFilaRecepcao =
nTamItemFilaRecepcao;
```

```
            pxParamRx->szBufferRx = pvPortMalloc( pxParamRx-
>nTamItemFilaRecepcao * sizeof( char ) );
            pxParamRx->xColaRecepcao = xColaRecepcao;
            pxParamRx->stTamanhoRx = 0;

            pxParamRx->CallbackFilaRecepcaoCheia =
callbackFilaRecepcaoCheia;
            pxParamRx->nTamItemFilaRecepcaoCheia =
nTamItemFilaRecepcaoCheia;

            pxParamRx->xColaRecepcaoCaracteres =
xQueueCreate(nQtdItensFilaRecepcao * nTamItemFilaRecepcao,
sizeof( char ) );
            vQueueAddToRegistry( pxParamRx-
>xColaRecepcaoCaracteres, "AppRxCarac");


            pxParamTx->nTamItemFilaEnvio = nTamItemFilaEnvio;
            pxParamTx->szBufferTx = pvPortMalloc( pxParamTx-
>nTamItemFilaEnvio * sizeof( char ) );
            pxParamTx->xColaEnvio = xColaEnvio;

            pxParamTx->paramRx = pxParamRx;

            pxParamTx->xColaEnvioCaracteres =
xQueueCreate(nQtdItensFilaEnvio * nTamItemFilaEnvio, sizeof(
char ) );
            vQueueAddToRegistry( pxParamTx-
>xColaEnvioCaracteres, "AppTxCarac");


            if(pxParamTx->xColaEnvio) {
           log("Configurando fila e task de transmissao...");
           //fila que passa em caracteres o que recebe em linhas
           serial_setFilaEnvio(pxParamTx->xColaEnvioCaracteres);

           xTaskCreate(taskSerialTxFunc,
                TASKSERIAL_TASTX_NOME,
           TASKSERIAL_TASTX_STACK_SIZE,
           (void *)pxParamTx,
           TASKSERIAL_TASKTX_PRIORITY,
           &xTaskHandle );

           }

           if(pxParamRx->xColaRecepcao) {
           //a recepção usa uma fila interna para caracteres que
vai aglutinando e enviando na fila
           // gxColaRecepcao
           //log("Configurando fila e task de recepcao...");
           serial_setFilaRecepcao(pxParamRx-
>xColaRecepcaoCaracteres);
```

```
          xTaskCreate(taskSerialRxFunc,
                TASKSERIAL_TASRX_NOME,
          TASKSERIAL_TASRX_STACK_SIZE,
          (void *)pxParamRx,
          TASKSERIAL_TASKRX_PRIORITY,
          NULL);
          }

          //time de disparo unico para terminar Rx por timeout
          /* gxTimerRecepcao =
xTimerCreate("TimerRecepcaoSerial",
          nTimeoutEntreCaracteresMs / portTICK_PERIOD_MS,
//convete para ticks
          pdFALSE, //sem autoreload (one shot)
          0, //timer ID
          callbackTimeoutEntreCaracteresFunc );
          */


          //inicia o hardware
          //log("Iniciando hardware serial...");
          serial_init();

          //log("taskserial_create() - fim da funcao...");

          return xTaskHandle;

          }


          void
sinalizaErroBufferCheio(param_dados_taskserial_rx_t *pxParamRx)
          {
          if(!pxParamRx->CallbackFilaRecepcaoCheia)
          {
          return;
          }

          //char *szBufferErro = pvPortMalloc(
gnTamItemFilaEnvio * sizeof( char ) );
          char szBufferErro[pxParamRx-
>nTamItemFilaRecepcaoCheia];


          pxParamRx->CallbackFilaRecepcaoCheia(
          utilmem_copiarTextoConst(szBufferErro,
          "{\"erro\": \"Erro ao adicionar item a
gxColaRecepcao. Texto: '"
          , pxParamRx->nTamItemFilaRecepcaoCheia));

          pxParamRx->CallbackFilaRecepcaoCheia(pxParamRx-
>szBufferRx);
```

```
          pxParamRx->CallbackFilaRecepcaoCheia(
          utilmem_copiarTextoConst(szBufferErro,
          "'\" }\r\n"
          , pxParamRx->nTamItemFilaRecepcaoCheia));

          }

          /*
           * @brief Código da task propriamente dito (para Rx)
           * @param void *pvParameters - Não uso
           * @retval None
           */
          void taskSerialRxFunc( void *pvParameters  )
          {
          param_dados_taskserial_rx_t *pxParamRx = NULL;

          pxParamRx = pvPortMalloc(
sizeof(param_dados_taskserial_rx_t) );

          utilmem_copiaMemoriaSecaoCritica((void *)pxParamRx,
pvParameters, sizeof(param_dados_taskserial_rx_t));

          char caracter;
          BaseType_t bCaracterQuebra;

          log("taskSerialRxFunc() executou");
          for (;;)
          {
            //espera com a task parada até chegar algo na fila
            if(xQueueReceive( pxParamRx-
>xColaRecepcaoCaracteres, (void *)&caracter, portMAX_DELAY) ==
pdPASS)
            {
            //para a temporização recarregar e voltar a esperar
otempo
          // testa depois -->   xTimerStart( gxTimerRecepcao, 0
);
          /*

          /*   char s[2];
          s[0] = caracter;
          s[1] = '\0';
          logvals("{", s);
          log("}");
          */
            //avalia \r e \n
            bCaracterQuebra = caracter == '\r' || caracter ==
'\n' || !caracter;

            if(bCaracterQuebra)
```

```
            {
            if(!pxParamRx->stTamanhoRx) continue;
            }

               if( bCaracterQuebra || (pxParamRx->stTamanhoRx
== (pxParamRx->nTamItemFilaRecepcao-1)) )
               {
             //Agora inicia denovo só ao receber um caractere
          // testa depois -->     xTimerStop( gxTimerRecepcao,
0 );


          if(!bCaracterQuebra)
          {
          pxParamRx->szBufferRx[(pxParamRx->stTamanhoRx)++] =
caracter;
          }
               pxParamRx->szBufferRx[pxParamRx->stTamanhoRx] =
0;
               pxParamRx->stTamanhoRx = 0;

               if(pxParamRx->xColaRecepcao)
               {
               //logvals("Encaminhado: ", pxParamRx-
>szBufferRx);

               //v0,3 - deixa de trava aqui, apenas envia
alerta pro callback de texto
               if(xQueueSendToBack(pxParamRx->xColaRecepcao,
(void *)pxParamRx->szBufferRx, 0) != pdTRUE)
               {
          sinalizaErroBufferCheio(pxParamRx) ;
               }
               }
               }
               else
               pxParamRx->szBufferRx[(pxParamRx-
>stTamanhoRx)++] = caracter;

              }
             }
          }

          /*
           * @brief Código da task propriamente dito (para Tx)
           * @param void *pvParameters - Não uso
           * @retval None
           */
          void taskSerialTxFunc( void *pvParameters )
          {
```

```
          param_dados_takserial_tx_t *pxParamTx = NULL;
          BaseType_t i;
          //BaseType_t bPrimeiro;
          int bPurgeRx;

             pxParamTx = pvPortMalloc(
sizeof(param_dados_takserial_tx_t) );

          utilmem_copiaMemoriaSecaoCritica((void *)pxParamTx,
pvParameters, sizeof(param_dados_takserial_tx_t));


          for (;;)
          {
            //converte Tx de linha para caracateres (de segundo
em segundo, pega...)
          if(xQueueReceive(pxParamTx->xColaEnvio, (void
*)pxParamTx->szBufferTx,
            /*1000 / portTICK_PERIOD_MS*/ portMAX_DELAY) ==
pdPASS)
          {

          taskENTER_CRITICAL();
          bPurgeRx = strcmp(pxParamTx->szBufferTx,
"cmd:reset_rx") == 0;
          taskEXIT_CRITICAL();
          if(bPurgeRx)
          {
          xQueueReset(pxParamTx->paramRx-
>xColaRecepcaoCaracteres);
          xQueueReset(pxParamTx->paramRx->xColaRecepcao);
          continue;
          }

          ////////////////////// usando buffer de tx
//////////////////////

          while(serial_testarEstaTransmitindo())
          {
          vTaskDelay(100 / portTICK_PERIOD_MS);
          }

          i = utilmem_comprimentoTextoSecaoCritica(pxParamTx-
>szBufferTx);

          if(i > pxParamTx->nTamItemFilaEnvio)
          {
          i = pxParamTx->nTamItemFilaEnvio;
          }

          serial_inicializaEnvio(pxParamTx->szBufferTx, i);
```

```
          /*
          bPrimeiro = uxQueueMessagesWaiting(pxParamTx-
>xColaEnvioCaracteres) == 0;

          for(i=0; pxParamTx->szBufferTx[i] && (i<pxParamTx-
>nTamItemFilaEnvio); i++){
          //trava a nivel de caracteres se a interrpção não
pegar a tempo tudo
          //nota que como está o Tx pode trabar o Rx
          if(xQueueSendToBack(pxParamTx->xColaEnvioCaracteres,
(void *)&pxParamTx->szBufferTx[i], 0) != pdPASS)
          {
          serial_habilitaEnvio();
          xQueueSendToBack(pxParamTx->xColaEnvioCaracteres,
(void *)&pxParamTx->szBufferTx[i], portMAX_DELAY);
          }
          }
          //força envio do primeiro caracteres por fora da fila
          if(bPrimeiro)
             serial_habilitaEnvio(); //poderia tocar isso po
uma task e mserial que
             //espera algo vir para a fila ou um evento para
avidar
          */

          }
          //else
          //{
          // serial_copiarRxParaFila();
          //}
            }
          }

          /*
           * @brief função de callback para o timeout entre
caracteres
           * @param pvParameters
           * @retval TaskHandle_t
           */
          void callbackTimeoutEntreCaracteresFunc(
TimerHandle_t xTimer __attribute__((unused)))
          {
          /*
          gszBufferRx[gstTamanhoRx] = 0;
          gstTamanhoRx = 0;

          if(gxColaRecepcao)
```

```
          {
          xQueueSendToBack(gxColaRecepcao, (void *)gszBufferRx,
portMAX_DELAY);
          }
          */
          }
```

task_serial.h

```
/*
 * task_serial.h
 *
 *  Created on: 18 de abr de 2018
 *      Author: Max
 */

#ifndef APLICACAO_TASKS_TASK_SERIAL_H_
#define APLICACAO_TASKS_TASK_SERIAL_H_

#ifdef ARDUINO
#include <Arduino_FreeRTOS.h>
#else
#include <FreeRTOS.h>
#endif

#include <queue.h>
#include <task.h>
#include "global.h"

//Tamanho de cada item de texto Recebido pela serial
(incluindo o 0 adicionado como termonador)
#define TASKSERIAL_MAXTAMLINHA 65


/*
 * @brief Cria a task
 * @param QueueHandle_t xColaEnvio - Fila de onde dem
os textos para envio
 * @param size_t nTamItemFilaEnvio - Tamanho do item
da fila de envio
 * @param size_t nQtdItensFilaEnvio - Quantos itens
tem a fila de envio
 * @param QueueHandle_t xColaRecepcao - Fila onde
deve colocar o texto recebido
 * @param size_t nTamItemFilaRecepcao - Tamanho do
item da fila de recepcao
 * @param size_t nQtdItensFilaRecepcao - Quantos
itens tem a fila de recepcao
 * @param uint32_t nTimeoutEntreCaracteresMs - tempo
em milesegundos entre caracteres
 * e Rx. Se pasa deste tempo termina a linha e envia
pela fila informada em xColaRecepcao
 * @param callbackEnvioTexto_t
callbackFilaRecepcaoCheia
 * @param size_t nTamItemFilaRecepcaoCheia - Tamanho
do item da fila de envio
 * @retval TaskHandle_t - Task de envio de dados
```

```
         */
        #ifndef _TASKSERIAL_C_
        extern
        #endif
        TaskHandle_t taskserial_create(
        QueueHandle_t xColaEnvio,
        size_t nTamItemFilaEnvio,
        size_t nQtdItensFilaEnvio,

        QueueHandle_t xColaRecepcao,
        size_t nTamItemFilaRecepcao,
        size_t nQtdItensFilaRecepcao,

        uint32_t nTimeoutEntreCaracteresMs,
callbackEnvioTexto_t callbackFilaRecepcaoCheia,
        size_t nTamItemFilaRecepcaoCheia);


        /*
         * @brief Define callback para enviar texto de debug
infomando que a fila de recepção está cheia;
         * Neste caso o pacote de texto é perdido.
         * @param callbackEnvioTexto_t
callbackFilaRecepcaoCheia
         */
        //void taskserial_setCallBackRecepcaoCheia(
callbackEnvioTexto_t callbackFilaRecepcaoCheia);

        /*
         * @brief Cria Envia texto, colocando o dado na fila
definida em taskserial_create.
         * Se for maior que TASKSERIAL_MAXTAMLINHA, trunca.
         * @param char *szTexto - Ponteiro apra texto a ser
enviado
         * @retval BaseType_t - (verdadeiro ou false). False
se a fila estiver cheia (não bloqueia)
         */
        #ifndef _TASKSERIAL_C_
        extern
        #endif
        BaseType_t taskserial_enviatexto(char *szTexto);


        /*
         * @brief Retorna a fila de envio
         * @param None
         * @retval QueueHandle_t - Fila setada ao criar (pode
ser NULL).
         */
        //QueueHandle_t taskserial_getFilaEnvio(void);
```

```
          /*
           * @brief Retorna a fila de recepção
           * @param None
           * @retval QueueHandle_t - Fila setada ao criar (pode
ser NULL).
           */
          //QueueHandle_t taskserial_getFilaRecepcao(void);

          #endif /* APLICACAO_TASKS_TASK_SERIAL_H_ */
```

util_memory.c

```
          /*
           * util_memory.c
           *
           *  Created on: 25 de abr de 2018
           *      Author: Max
           */

          #define _UTILMEM_C_

          #include "util_memory.h"

          #ifdef ARDUINO
          #include <Arduino_FreeRTOS.h>
          #else
          #include <FreeRTOS.h>
          #endif

          #include <task.h>
          #include <string.h>


          /*
           * @brief Determina, byte a byte, o tamanho de um
texto (até caracter nulo, não incluso),  dentro de uma seção
critica
          */
          size_t utilmem_comprimentoTextoSecaoCritica(char
*szTexto)
          {
          size_t nRet = 0;

          taskENTER_CRITICAL();

          while(*szTexto++)
          nRet++;

          taskEXIT_CRITICAL();

          return nRet;
          }

          /*
           * @brief Determina, byte a byte, o tamanho de um
texto (até caracter nulo, não incluso),  dentro de uma seção
critica
          */
          size_t
utilmem_comprimentoTextoSecaoCriticaConst(const char *szTexto)
          {
          size_t nRet = 0;
```

```
          taskENTER_CRITICAL();

          while(*szTexto++)
          nRet++;

          taskEXIT_CRITICAL();

          return nRet;
          }

          /*
           * @brief Determina, byte a byte, o tamanho de um
texto (até caracter nulo, não incluso),  dentro de uma seção
critica
          */
          size_t
utilmem_comprimentoTextoSecaoCriticaVolatile(volatile char
*szTexto)
          {
          size_t nRet = 0;

          taskENTER_CRITICAL();

          while(*szTexto++)
          nRet++;

          taskEXIT_CRITICAL();

          return nRet;
          }


          /*
           * @brief Copia, byte a byte,  dentro de uam seção
critica
          */
          void utilmem_copiaMemoriaSecaoCritica(void
*pvDestino, void *pvOrigem, size_t nTam)
          {
          taskENTER_CRITICAL();

          char *pucDest = (char *)pvDestino;
          char *pucOrig = (char *)pvOrigem;

          while(nTam--)
          *pucDest++ = *pucOrig++;

          taskEXIT_CRITICAL();

          }
```

```
          /*
           * @brief Copia texto para buffer, avaliando o menos
tamanho (considera texto com terminador nulo na origem)
           * e pega seu tamanho baseado nisto
           */
          char * utilmem_copiarTextoMenorTamanho(char
*szDestino, char *szOrigem, size_t nTamBufferDestino)
          {

            size_t nTamStrParam = szOrigem ? strlen(szOrigem)+1
: 0;
            size_t nTamCopia = nTamBufferDestino < nTamStrParam
? nTamBufferDestino : nTamStrParam;

             szDestino[0] = '\0';
             if(szOrigem)
             {
             //memcpy(szDestino, szOrigem, nTamCopia);
             utilmem_copiaMemoriaSecaoCritica(szDestino,
szOrigem, nTamCopia);
             }
             szDestino[nTamBufferDestino-1] = 0; //garante que
termina a string

             return szDestino;
          }


          /*
           * @brief Copia texto para buffer, respeitando o
tamango máximo, evitando
           * falhar de código.
           * Algoritimo ajustado apra o caso do buffe de origem
e destino serem os mesmos
           */
          char * utilmem_copiarTexto(char *szDestino, char
*szOrigem, size_t nTamBufferDestino)
          {
          char *szRet = szDestino;

          if(szOrigem == szDestino)
          {
          szDestino[nTamBufferDestino-1] = '\0';
          return szRet;
          }

          *szDestino = '\0';
          while(*szOrigem)
          {
          *szDestino = *szOrigem;
```

```
szOrigem++;
szDestino++;
*szDestino = '\0';

nTamBufferDestino--;

if(nTamBufferDestino == 1)
{
*szDestino = '\0';
break;
}
}
return szRet;
}


/*
 * @brief Copia texto para buffer, respeitando o
tamango máximo, evitando
 * falhar de código.
 */
char * utilmem_copiarTextoConst(char *szDestino,
const char *szOrigem, size_t nTamBufferDestino)
{
char *szRet = szDestino;
*szDestino = '\0';
while(*szOrigem)
{
*szDestino = *szOrigem;
szOrigem++;
szDestino++;
*szDestino = '\0';

nTamBufferDestino--;

if(nTamBufferDestino == 1)
{
*szDestino = '\0';
break;
}
}
return szRet;
}
```

util_memory.h

```
/*
 * util_memory.h
 *
 *  Created on: 25 de abr de 2018
 *      Author: Max
 */


#ifndef APLICACAO_UTIL_MEMORY_H_
#define APLICACAO_UTIL_MEMORY_H_

#include <stddef.h>


/*
 * @brief Determina, byte a byte, o tamanho de u
mtexto (até caracter nulo, não incluso),  dentro de uma seção
critica
*/
#ifndef _UTILMEM_C_
extern
#endif
size_t utilmem_comprimentoTextoSecaoCritica(char
*szTexto);

/*
 * @brief Determina, byte a byte, o tamanho de u
mtexto (até caracter nulo, não incluso),  dentro de uma seção
critica
*/
#ifndef _UTILMEM_C_
extern
#endif
size_t
utilmem_comprimentoTextoSecaoCriticaConst(const char *szTexto);

/*
 * @brief Determina, byte a byte, o tamanho de u
mtexto (até caracter nulo, não incluso),  dentro de uma seção
critica
*/
#ifndef _UTILMEM_C_
extern
#endif
size_t
utilmem_comprimentoTextoSecaoCriticaVolatile(volatile char
*szTexto);
```

```
          /*
           * @brief Copia, byte a byte,  dentro de uma seção
critica
          */
          #ifndef _UTILMEM_C_
          extern
          #endif
          void utilmem_copiaMemoriaSecaoCritica(void
*pvDestino, void *pvOrigem, size_t nTam);

          /*
           * @brief Copia texto para buffer, avaliando o menos
tamanho (considera texto com terminador nulo na origem)
           * e pega seu tamanho baseado nisto
           */
          #ifndef _UTILMEM_C_
          extern
          #endif
          char * utilmem_copiarTextoMenorTamanho(char
*szDestino, char *szOrigem, size_t nTamBufferDestino);

          /*
           * @brief Copia texto para buffer, respeitando o
tamango máximo, evitando
           * falhar de código.
           */
          #ifndef _UTILMEM_C_
          extern
          #endif
          char * utilmem_copiarTexto(char *szDestino, char
*szOrigem, size_t nTamBufferDestino);

          /*
           * @brief Copia texto para buffer, respeitando o
tamango máximo, evitando
           * falhar de código.
           */
          #ifndef _UTILMEM_C_
          extern
          #endif
          char * utilmem_copiarTextoConst(char *scDestino,
const char *szOrigem, size_t nTamBufferDestino);


          #endif
```

Apêndice E – Código do código PHP

Segue corpo do script logdispositivo.php. Você pode configurar um servidor HTTP com suporte ao php e alterar o arquivo **\src\aplicacao\global.h** do projeto em C da aplicação, para apontar para o servidor e a variável $xxx do script para indicar a pasta real da base de arquivos do ser servidor.

A alteração do arquivo ccccc consiste em alterar os seguintes define:

```
#define CLIENTEHTTP_REDE "tccmaxback"
#define CLIENTEHTTP_SENHA "12345678"
#define CLIENTEHTTP_HOST "192.168.43.180"
#define CLIENTEHTTP_PORTA "8080"

 #define CLIENTEHTTP_DOCUMENTO_GET "/v0/logdispositivo.php?ID=100&i=0&t="
```

O primeiro define o nome da rede sem fio e o segundo a senha.

No terceiro parâmetro temos o endereço IP v4 do computador odne roda o script PHP e no quarto a parta (no exemplo 8080).

No quinto parâmetro (CLIENTEHTTP_DOCUMENTO_GET) temos a string que deve ser usada para formar a URL de envio do coamndo GET. A esta string será anexado ainda o texto do dado a ser efetivamente enviado.

Acabamos trabalhando com um ID fico com 100 e i, que é uma espécie de sequencial de leitura, fica fixo em 0.

Se você fosse fazer uma aplicação real ,seria conveniente uma interface de terminal serial para configurar o ID e depois usar a EEPROM ou outra memóeria não volátil para gravar e ler esta configuração.

Já a alteração no arquivo PHP consiste apenas em alterar a variável $caminho_base_real para o caminho base, que no meu caso está no padrão do windows (com “\”) .

```
$caminho_base_real = "C:\\MAXBACK\\dados_onedrive\\OneDrive\\Projetos\\Livro Uso de freeRTOS como base para programação multiplataforma\\fonte php\\";
```

Segue o arquivo d oscript. Ele nã ofaz nada mais que definir um nome de arquivo por ID e por dia e salvar as linhas recebidas naquele arquivo.

\vo\logdispositivo.php

```
          <?php

          //http://localhost:8080/v0/logdispositivo.php?
ID=1&i=0&t=Teste!&caminho=a&mascara=b


          //PASTA BASE DSO SCRIPTS
          $caminho_base_real =
"C:\\MAXBACK\\dados_onedrive\\OneDrive\\Projetos\\Livro Uso de
freeRTOS como base para programação multiplataforma\\fonte
php\\";


          date_default_timezone_set('America/Sao_Paulo');

          //CONFIGURAÇÃO PARA O SCRIPT USAR
          $config = array("caminho" => $caminho_base_real .
"dados",
                          "mascara" => "LogImp-ID%ID%-
Data%ano%%mes%%dia%.txt",
                          "data" =>  getdate(),
                          "salva_arquivo" => false
          );

          $resultado = array("cdCapturaIni" => 0,
                             "cdCapturaFim" => 0,
                             "cdIndice" => 0);


          function valorParametro(string $nome): string
          {
            //echo "[Lendo param ".$nome."]<br>";
              return $_GET[$nome] ?? $_POST[$nome] ?? "";
          }

          function valorConfig(string $nome, array $config)
          {
            //echo "[Lendo param ".$nome.":".
($config_adicional[$nome]?? "") . "]<br>";
              return $config[$nome] ?? "";
          }

          function validarParametros($config)
          {
              if(valorParametro('ID') == "")
              {
                 throw new Exception("Parâmetro 'ID' não
informado");
              }
```

```
            if(valorParametro('i') == "")
            {
               throw new Exception("Parâmetro 'i' não
informado");
            }

            if(valorConfig('caminho', $config) == "")
            {
               throw new Exception("Parâmetro de ambiente
'caminho' não definido");
            }

            if(valorConfig('mascara', $config) == "")
            {
               throw new Exception("Parâmetro de ambiente
'mascara' não definido");
            }
          }

          function DefinirCaminhoArq($config, $ID, $i):string
          {
            $sNomeArq = "";
            $sCaminhoEmDisco = "";
            $sMascaNomeArquivo = "";
            $agora = valorConfig("data", $config);

            $sCaminhoEmDisco = valorConfig('caminho', $config);
            $sMascaNomeArquivo = valorConfig('mascara',
$config);


            $dia = str_pad($agora["mday"], 2, "0",
STR_PAD_LEFT);
            $mes = str_pad($agora["mon"], 2, "0", STR_PAD_LEFT);
            $ano = str_pad($agora["year"], 4, "0",
STR_PAD_LEFT);

            //echo "[$dia][$mes][$ano]";

            $sNomeArq = str_replace('%dia%', $dia,
$sMascaNomeArquivo);
            $sNomeArq = str_replace('%mes%', $mes, $sNomeArq);
            $sNomeArq = str_replace('%ano%', $ano, $sNomeArq);
            $sNomeArq = str_replace('%ID%', $ID, $sNomeArq);
            $sNomeArq = str_replace('%i%', $i, $sNomeArq);

            $sCaminhoArq = $sCaminhoEmDisco . '\\' . $sNomeArq;
            $sCaminhoArq = str_replace("\\\\", "\\",
$sCaminhoArq);

            return $sCaminhoArq;
```

```
            }

            function abre($sCaminhoArq)
            {

               if(file_exists($sCaminhoArq))
                 return fopen($sCaminhoArq, "a");

               return fopen($sCaminhoArq, "w");
            }

            function salvaNovaCaptura($data, $ID, $i, $t):array
            {
            global $config;

            $config["data"] = $data;

               $sCaminhoArq = DefinirCaminhoArq($config, $ID,
$i);

               $s = "";
               $sDadosAdicionais = "";
               $i = 0;

               $F = abre($sCaminhoArq);

               $slDados = explode("\n", $t);

            $t = "";

            foreach($slDados as $s)
               {
                  $sDadosAdicionais = "";

                  $s = trim($s);

                  $adddata = (strpos(strtolower($s), "data :") !==
false) ||
                         (strpos(strtolower($s), "data:") !==
false) ||
                         (strpos(strtolower($s), "*** teste
comuninicacao ***") !== false);
                  $addhora = (strpos(strtolower($s), "hora :") !==
false) ||
                         (strpos(strtolower($s), "hora:") !==
false) ||
                         (strpos(strtolower($s), "*** teste
comuninicacao ***") !== false);
                  $agora = valorConfig("data", $config);
```

```
                $dias = str_pad($agora["mday"], 2, "0",
STR_PAD_LEFT);
                $mess = str_pad($agora["mon"], 2, "0",
STR_PAD_LEFT);
                $anos = str_pad($agora["year"], 4, "0",
STR_PAD_LEFT);
                $horass = str_pad($agora["hours"], 2, "0",
STR_PAD_LEFT);
                $minutoss = str_pad($agora["minutes"], 2, "0",
STR_PAD_LEFT);

                if( $adddata && $addhora)
                {
                   $sDadosAdicionais .= "-  Data e Hora do
servidor: " . "$dias/$mess/$anos - $horass:$minutoss";
                }
                else
                if( $adddata  )
                {
                   $sDadosAdicionais .= "-  Data servidor:
$dias/$mess/$anos";
                }
                else if( $addhora )
                {
                   $sDadosAdicionais .= "-  Hora servidor: " .
"$horass:$minutoss";
                }

                $sDadosAdicionais .= "Pacote: " . $s;

                if(strpos($s, "GC:F,") !== false)
                {
                   $dic = array("GC:F,0;" => "Giroscopio: Tipo
filtro: 0 (gtfSemFiltro)",
                                "GC:F,1;" => "Giroscopio: Tipo
filtro: 1 (gtfMinimoValorAlgumEixo)",
                                "GC:F,2;" => "Giroscopio: Tipo
filtro: 2 (gtfEliminaAbaixoDoMinimoValor)",
                                "GC:F,3;" => "Giroscopio: Tipo
filtro: 3 (gtfMinimoValorTodosEixo)",
                                "GC:F,4;" => "Giroscopio: Tipo
filtro: 4 (gtfMinimoValorEixoX)",
                                "GC:F,5;" => "Giroscopio: Tipo
filtro: 5 (gtfApenasValorEixoX)",
                                "GC:F,6;" => "Giroscopio: Tipo
filtro: 6 (gtfMinimoValorEixoY)",
                                "GC:F,7;" => "Giroscopio: Tipo
filtro: 7 (gtfApenasValorEixoY)",
```

```
                                "GC:F,8;" => "Giroscopio: Tipo
filtro: 8 (gtfMinimoValorEixoZ)",
                                "GC:F,9;" => "Giroscopio: Tipo
filtro: 9 (gtfApenasValorEixoZ)",
                                "GC:F,10;" => "Giroscopio: Tipo
filtro: 10 (gtfUsarFuncaoFiltro)"
                    );

                    $s2 = str_replace(":", ",", $s);
                    $s2 = str_replace(";", ",", $s2);

                    $campos = explode(",", $s2);

                    $dado_cru = "Giroscopio: Tipo filtro: " .
$campos[2] . " (desconhecido)";
                    $s = $dic[$s] ?? $dado_cru;
                }
                //"BA:0;"
                else if(strpos($s, "BA:") !== false)
                {
                    $s = str_replace("BA:", "Botao ", $s);
                    $s = str_replace(";", "", $s) . ":
pressionado";
                }
                //"SI:;"
                else if(strpos($s, "SI:;") !== false)
                {
                    $s = "Msg: \"Sistema iniciado\"";
                }
                //"GL:%s,%s,%s;"
                else if(strpos($s, "GL:") !== false)
                {
                    $s = str_replace(":", ",", $s);
                    $s = str_replace(";", ",", $s);

                    $campos = explode(",", $s);

                    $s = sprintf("Giroscopio: x:%s, y:%s, z:%s",
$campos[1], $campos[2], $campos[3]);
                }
                //"JL:%s,%s;"
                else if(strpos($s, "JL:") !== false)
                {
                    $s = str_replace(":", ",", $s);
                    $s = str_replace(";", ",", $s);

                    $campos = explode(",", $s);

                    $s = sprintf("Joystick: x:%s, y:%s",
$campos[1], $campos[2]);
```

```
                }
                //"MN:n,N,T;" <- teste em partes
                else if(strpos($s, "MN:") !== false)
                {
                   $s = str_replace(":", ",", $s);
                   $s = str_replace(";", ",", $s);

                   $campos = explode(",", $s);

                   $parcial = "";
                   $indice = intval($campos[1]);
                   $qtd = intval($campos[2]);

                   if($qtd != 1)
                   {
                      $parcial = sprintf(" (%d/%d)", $indice+1,
$qtd);
                   }

                   $s = sprintf("Msg: \"%s\" %s", $campos[3],
$parcial);
                }

                $dias = str_pad($agora["mday"], 2, "0",
STR_PAD_LEFT);
                $mess = str_pad($agora["mon"], 2, "0",
STR_PAD_LEFT);
                $anos = str_pad($agora["year"], 4, "0",
STR_PAD_LEFT);
                $horass = str_pad($agora["hours"], 2, "0",
STR_PAD_LEFT);
                $minutoss = str_pad($agora["minutes"], 2, "0",
STR_PAD_LEFT);
                $segundoss = str_pad($agora["seconds"], 2, "0",
STR_PAD_LEFT);

                $s = "[$horass:$minutoss:$segundoss]: " . $s;

                if($sDadosAdicionais != "" && strpos($s,
"@@dadosextrasservidor:"))
                   $t = $s . "@@dadosextrasservidor:" .
$sDadosAdicionais;
                else
                   $t = $s;

          if($s != "" || $sDadosAdicionais != "")
          {
            fwrite($F, $t . chr(13). chr(10));
          }
```

```
            }
            fclose($F);
         }

          try
          {
             validarParametros($config);
          }
          catch (Error $e)
          {
             echo "N,-1:Ocorreu um erro: " . $e->getMessage();
          }
          salvaNovaCaptura(getdate(), valorParametro('ID'),
valorParametro('i'), valorParametro('t'));

     echo "S\r\n";
```

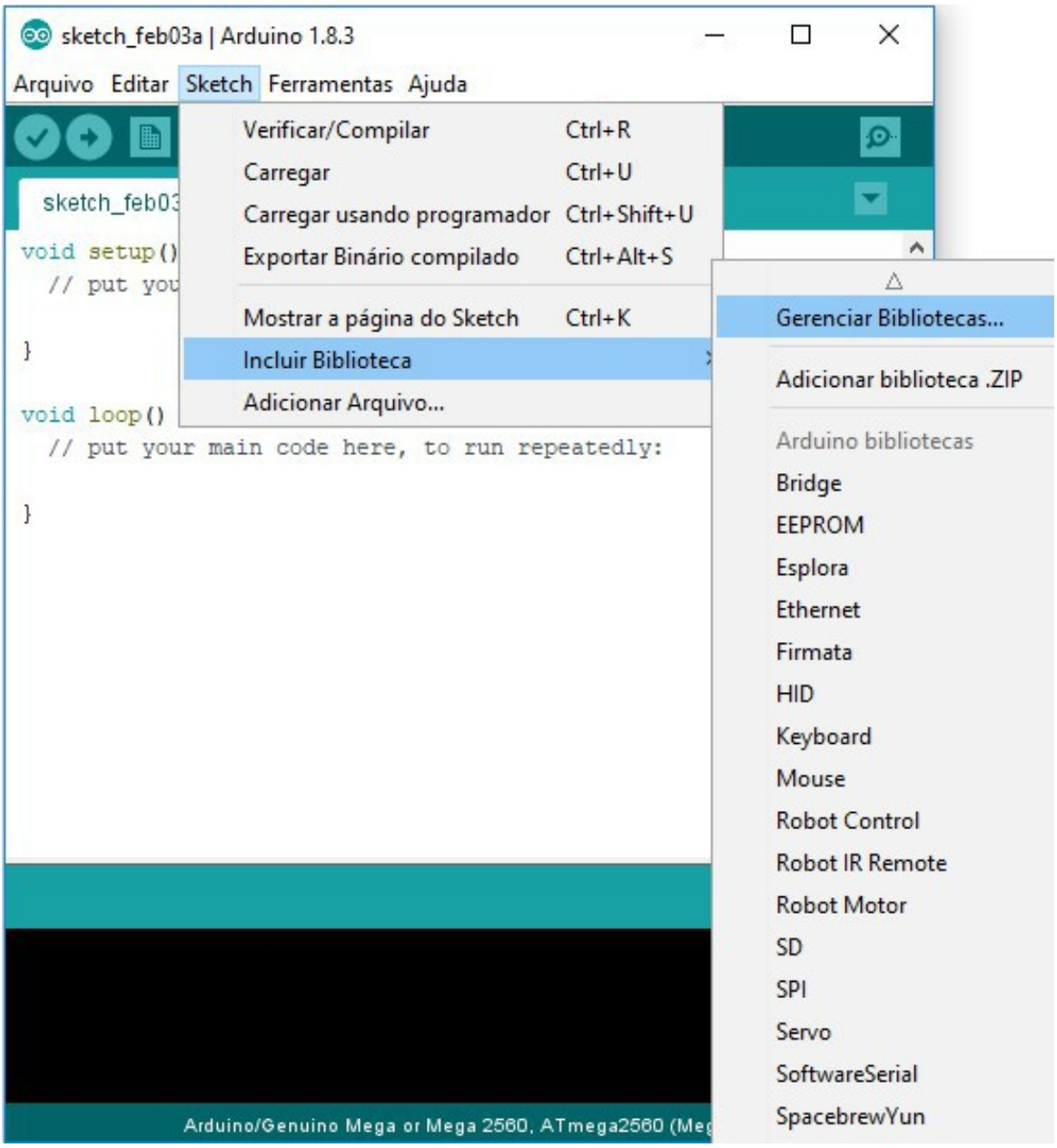
Apéndice F - Instalación de la biblioteca freeRTOS

Las siguientes son las instrucciones de instalación de freeRTOS:

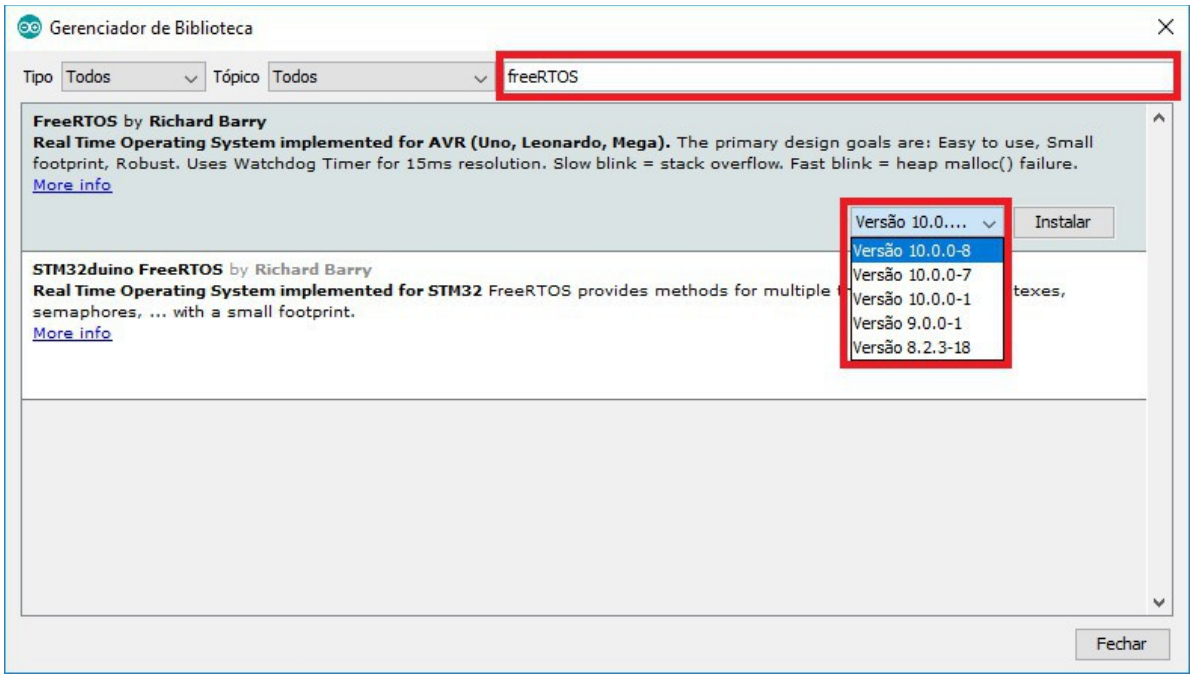
Comencemos instalando la biblioteca freeRTOS de Arduino. Debe estar conectado a Internet, ya que la biblioteca se descargará durante el proceso de instalación.

Consejo: Lea todos los pasos cuidadosamente antes de comenzar y luego úselos como guía si lo prefiere.

Paso 1: Abra Arduino y luego vaya al elemento de menú Boceto \ Agregar biblioteca \ Administrar bibliotecas ... como se muestra a continuación:

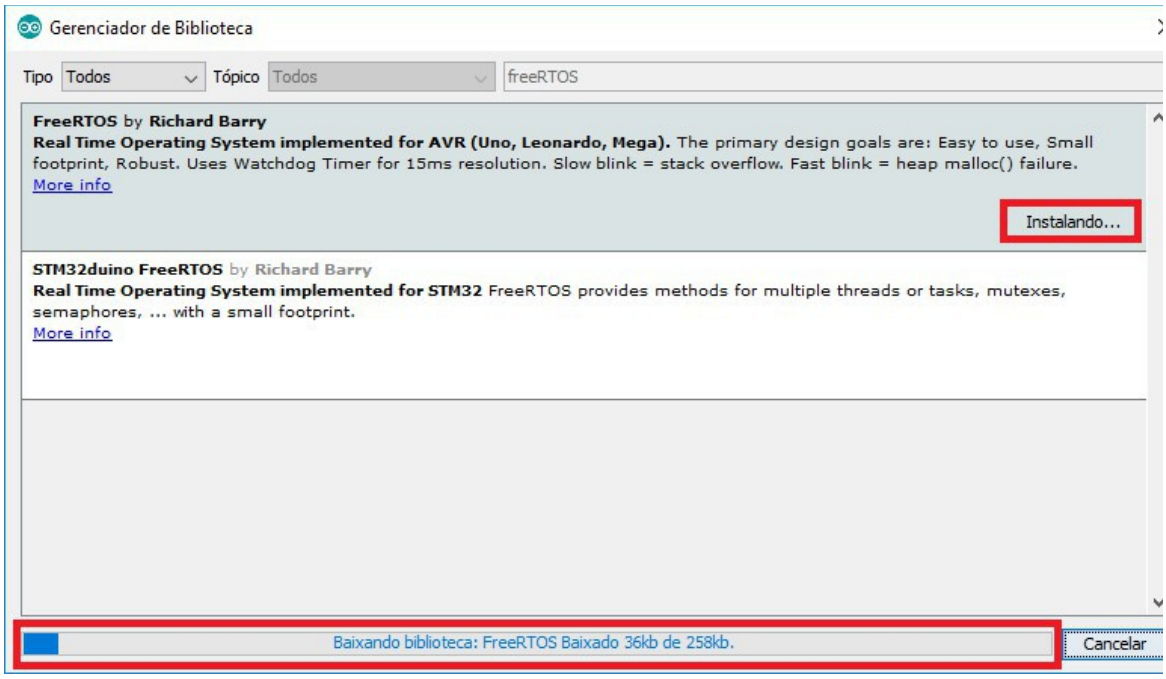


Paso 2: en el cuadro de texto para refinar su búsqueda, escriba freeRTOS (como se resalta en la imagen) y seleccione el elemento para AVR (Arduino Uno, Leonardo, Mega). Luego seleccione la última versión (casilla de verificación junto al botón de instalación).

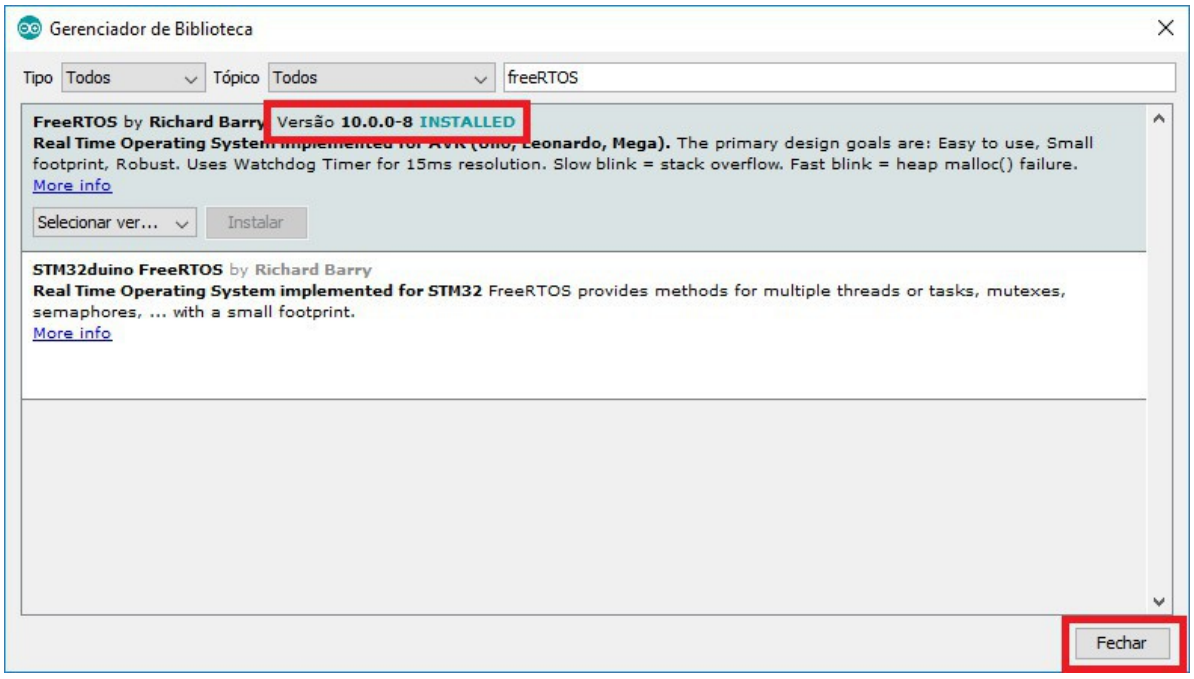


Atención : Todos los ejemplos en el libro se hicieron con la versión 10.0.0-8 de esta biblioteca.

Paso 3: Presione el botón "Instalar" y espere mientras se descarga la biblioteca.



Paso 4: Si todo va bien, se mostrará la información de la versión instalada. Simplemente presione el botón de cerrar.



Notas

[←1]

En la implementación del puerto FreeRTOS que se está trabajando en este libro, la función loop() se ha asociado con la función RTOS IDLE, es decir, se llama de vez en cuando. Este es un tema avanzado, por lo que podemos suponer por ahora que esta función no debería recibir ningún código.

[←2]

Pero puede suceder que la segunda *task* elimine elementos mientras que la primera es agregar elementos. Ejemplo: antes de que el elemento con valor 3 se ponga en cola, el elemento con valor 1 ya se ha eliminado.

[←3]

Como se comentó anteriormente, podría producirse un error debido a la falta de memoria. Este tipo de situación surge cuando comenzamos a hacer sistemas más grandes y la memoria disponible para la asignación dinámica es insuficiente.

[← 4]

En el Apéndice B - Notas del seminario web freeRTOS (Microgenios y embacados.com.br) transcribo mis notas de la buena explicación del profesor Fernando Simplício sobre este tema en un seminario web organizado por Microgenios y embarados.com.br en 2017.com el profesor Fernando Simplício. Puede ser útil para ti.

[←5]
Una posible solución a esto es utilizar el tiempo del software freeRTOS para un control de tiempo de espera (*timeout*).

[←6]

No será común en este libro proponer ejercicios, pero las colas son el mecanismo más importante de comunicación de *tasks* , y vale la pena detenerse y jugar con la comunicación de *tasks* y asegurarse de comprender cómo funciona y cómo funciona. qué pasa.

[←7]

Por supuesto, este es un ejemplo que estoy imaginando, ya que nunca he examinado el *software* de un televisor. También es posible que la implementación estuviera en una cola o en alguna otra estrategia de implementación. Siempre es importante evaluar el costo para el sistema y la prioridad de una *task* sobre otra, extrayendo así el mejor rendimiento posible para las actividades críticas y dejando a los demás desempeñándose de manera aceptable.

[←8]
Tenemos desde este lugar el código original, con términos en idioma portugués.

[←9]
"L" viene del portugués "Ligado" y "D" del portugués "Deligado”

[←10]
- Este proyecto fue adaptado para este libro, pero con muchas similitudes con el proyecto original, eliminando cualquier referencia a otra plataforma (STM32) o compilación condicional creada para permitir ajustes en la compilación de código común.

Si desea ver también la adaptación STM32, está disponible en el libro "freeRTOS como base para la programación multiplataforma: con Arduino y STM32": https://www.amazon.com.br/freeRTOS-como-base-programa%C3%A7%C3%A3o-multiplataforma-ebook/dp/B07KSKWWY9

[←11]

Ellos son: L3G4200D (Sensor de frecuencia angular de 3 ejes) - giroscopio de 3 ejes; ADXL345 (Acelerómetro digital de 3 ejes) - Acelerómetro de 3 ejes; HMC5883L (brújula digital de 3 ejes): brújula digital / magnetómetro; BMP085 (Sensor de presión barométrica): sensor de temperatura y presión. Para obtener más detalles, consulte la excelente tutoría de comunicación con este módulo disponible en https://www.arduinoecia.com.br/2014/09/sensor-gy-80-acelerometro-bussola-barometro.html . Acceso el 25/09/2018.

[←12]

Esta adaptación no es ideal en un proyecto profesional ya que es muy limitada. Los circuitos con transistores o la adaptación de señal CIS proporcionan mejores adaptaciones. Puede usar módulos convertidores de señal como este de RoboCore: https://www.robocore.net/loja/produtos/conversor-de-nivel-logico.html

[←13]

Esta es una vergüenza técnica que no resolví de manera oportuna. El entorno arduino puede ser un poco misterioso para aquellos diseñadores que están acostumbrados a la programación incrustada C pura, donde puede configurar exactamente cómo se construirán los módulos. Arduino tiende a tener módulos como bibliotecas y esconde detalles del usuario que pueden ser inesperados y no necesita preocuparse por dichos detalles. Este módulo se mantiene de todos modos, ya que permite agrupar el control de *timeout* de espera de las multas.

[←14]

Esta función es un *callback* de llamada definida al llamar a ExecutarSerial1_registrarCalbackSerialEvent() en serial_init() para permitir que la función serialEvet() de arduino advierta al módulo serial.c de los datos detectados en la función. Es el equivalente de la función que maneja la interrupción en serie en otras plataformas.

Table of Contents

Your gateway to knowledge and culture. Accessible for everyone.