Sistemas Embebidos

CONECTIVIDAD, TIEMPO REAL E INTERRUPCIONES

Practica 7

**Introducción**

En esta práctica se centra en el estudio y puesta en practica de los microcontroladores utilizando dos dispositivos distintos: el Arduino Nano IoT 33 y el ESP32 en su versión Lolin32. El Arduino Nano IoT 33 es un microcontrolador diseñado especialmente para aplicaciones de Internet de las Cosas, mientras que el ESP32 ofrece potentes capacidades de procesamiento y conectividad Wi-Fi y Bluetooth. Hasta ahora, se ha utilizado el Arduino Nano IoT para construir el primer dispositivo embebido, obteniendo datos de un sensor. Sin embargo, en esta ocasión, ampliará las capacidades del dispositivo al aprovechar su capacidad de conectividad. Por otro lado, explorará el ESP32 para entender su potencial en términos de programación en tiempo real, haciendo hincapié en el uso de interrupciones y los modos de bajo consumo disponibles en este microprocesador. Este enfoque dual permitirá adentrarnos en diferentes aspectos del desarrollo de sistemas embebidos y comprender mejor las posibilidades que ofrecen estos microcontroladores en diversas aplicaciones.

El dispositivo Arduino Nano 33 IoT que además las características habituales del Arduino Nano, incorpora conectividad WiFi y conectividad Bluetooth mediante un módulo Nina W102 uBlox, por lo que es ideal para proyectos IoT.

Imagen que contiene circuito, electrónica

Descripción generada automáticamente

Para poder programar esta placa con el Arduino IDE, tendremos que instalar mediante el Boards manager la familia de placas Arduino SAMD Boards (32-bits ARM Cortex-M0+). Como siempre también tendrá que seleccionar la placa Arduino NANO 33 IoT y el puerto del ordenador al que este conectada.

**Parte 1**

Antes de empezar a resolver la practica cabe indicar que no pondré pregunta a pregunta, si no que iré contestando a ellas de manera ordenada tal y como se indica en el pdf de la práctica.

En primer lugar tendrá que **configurar Ubidots Steam** para ello tendrá que crear una cuenta en la plataforma, accediendo al portal de esta, <https://stem.ubidots.com> .

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Una vez terminado el registro, tendrá que crear un dispositivo llamado "Nano-Iot-Test", para realizar esto tendrá que, desde el menú superior hacer clic en Dispositivos y en su página crear un dispositivo seleccionando en "**Create Device**".

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamenteSe abrirá un desplegable, selecciona dispositivo en blanco tal y como se indica en el enunciado. Y por último, completa cualquier otra información relevante que se requiera y haz clic para finalizar la creación del dispositivo.

Una vez dentro del dispositivo "Nano-Iot-Test", siga estos pasos para **crear la variable pulsaciones**, que se encargará de registrar las pulsaciones obtenidos por nuestro sensor.

Para crearla correctamente, tal y como se indica, tendrá que ser una ‘Raw Variable’

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

Tendrá que asignarle el nombre ‘pulsaciones’ y posteriormente establecer un rango permitido delimitado entre 0 y 220.

Una vez tenga la variable insertada en el device, podrá pasar a **modificar el Dashboard**, para ello utilice el predefinido y simplemente lo modifíquelo para añadir las siguientes widgets.

Tal y como se indica, una **widget de métrica** para mostrar el último valor de la variable ‘pulsaciones’, un **widget de grafico de línea** para representar los últimos valores de esta y por último uno **para mostrar una imagen de nuestro dispositivo**.

**Widget para ultimo valor pulsaciones Widget para la gráfica de pulsaciones**

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Una vez ya ha terminado de configurar Ubidots, ya podrá realizar la implementación tanto del código pertinente para conectar con Ubidots, como de la implementación del Arduino en nuestra placa.

Interfaz de usuario gráfica, Aplicación, Chat o mensaje de texto

Descripción generada automáticamentePrimer a obtendrá los credenciales de Ubidots que utilizará más tarde. Para ello desde el menú de Ubidots seleccione la opción **API Credentials** y copie el token de Ubidots, ya que este nos permitirá establecer conexión con la plataforma posteriormente.

Para la **conectividad WiFi**, haremos uso de la librería de Arduino Wi-FiNINA que deberá instalar previamente en el gestor de librerías del IDE.

Pantalla de un celular de un mensaje en letras blancas

Descripción generada automáticamente con confianza mediaPara los credenciales de la red WiFi, es una buena practica mantener los secretos y contraseñas separadas del código de producción. Para ello tendrá que crear otro archivo Arduino llamado secrets.h , cuyo contenido podría ser:

Comenzando con el programa principal, importaremos la libreria WiFi y también el archivo de secretos creado anteriormente.



Podrá pasar las variables del compilador en variables char[].

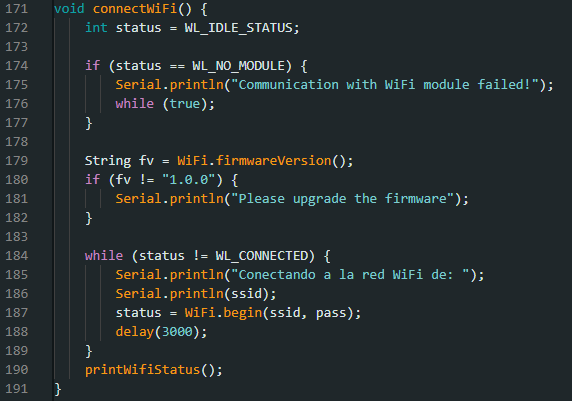


Aunque solo sea para conectar a la red WiFi no es necesario, es habitual crear una variable global de programa para el cliente WiFi que permitirá usar los protocolos TCP, UDP, DNS.

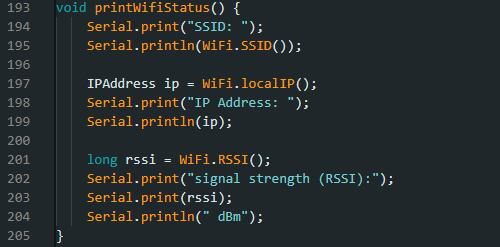


Esta misma variable luego podrá ser pasada a librerías que implementen otros protocolos como HTTP y MQTT. Esto se verá en los próximos apartados.

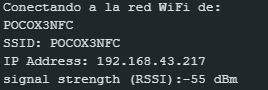
Antes de conectar a la red, tendrá que hacer algunas comprobaciones para asegurar que puede comunicarse correctamente con el chip WiFi y que la versión firmare es correcta. Para ello implemente el siguiente código.



Se realizará la conexión mediante la función WiFi.begin() que recibe como parámetros el nombre y contraseña de la red WiFi. Además esta se encuentra dentro de un bucle por su falla su intento de conexión para que lo vuelva a intentar.

Por último hay que comentar que la función printWifiStatus() imprime en el Serial algunas características de nuestra red WiFi si la conexión se ha efectuado exitosamente.

Ahora una vez implementado lo anterior debería de devolvernos lo siguiente:



Tal y como se pide en la practica a **enviar las pulsaciones a Ubidots mediante el protocolo HTTP**. Para ello añade a continuación todo lo relacionado con HTTP.

Para facilitar el uso del protocolo HTTP en el programa use la libreria **ArduinoHttpClient** que habrá instalar mediante el gestor de paquetes. A continuación importe la libreria y cree el cliente HTTP:

Con esta última linea, el cliente HTTP, obtendrá la red WiFi, para tener acceso a los protocolos TCP, UDP…, dispondrá del endpoint que posteriormente se indicará como establecerlo y por último, el puerto de conexión.

Ahora bien, para enviar la variable a la plataforma, según la documentación de Ubidots, para actualizar el valor de una variable habrá que hacer una petición POST al endpoint:

**https://industrial.api.ubidots.com/api/v1.6/variables/<variable id>/values**

Una vez conectados a una red WiFi, se podrá comenzar a hacer peticiones, para ello utilice las siguientes funciones manteniendo el orden:

* **httpClient.beginRequest()** Comenzamos una petición HTTP.
* **httpClient.get(url), httpClient.post()** Especificamos el endpoint, pasando un path absoluto como parámetro /api/v1.6/variables/pulsaciones/values.
* **httpClient.sendHeader(key, value)** Especifica headers. Este método es opcional y se puede repetir varias veces para especificar múltiples headers.
* **httpClient.beginBody()** Iniciamos la especificación del body de la petición.
* **httpClient.print(postData)** Añadimos el body. Esta función también puede usarse tantas veces como haga falta.
* **httpClient.endRequest()** Finalizamos y enviamos la petición.

Una vez hecho esto, el servidor debería recibir y responder a la petición. Una vez mandada la petición, usaremos las siguientes funciones para obtener la respuesta:

* **httpClient.responseStatusCode()**  Devuelve un int con el status de la petición HTTP.
* **httpClient.responseBody()**  Devuelve un String con la respuesta del servidor

Pantalla de computadora con letras

Descripción generada automáticamente con confianza mediaUna vez entendidas todas las funciones que se pueden utilizar solo falta implementar la función que se encargará de enviar la información.

Como se pide en la práctica, **observa los cambios obtenidos en el dashboard** una vez se envía la información desde nuestro Arduino a la plataforma de Ubidots a lo largo del tiempo.

Imagen que contiene electrónica, circuito

Descripción generada automáticamente

Observa como el Line Chart muestra la evolución de las pulsaciones obtenidas por el sensor a lo largo de las mediciones, como en el centro está la imagen del Arduino y por último el Last Value de la variable pulsaciones.

Una vez finalizado con el protocolo HTTP, se pide implementar el **protocolo MQTT**, para enviar y recibir datos. Para ello cree un nuevo botón en el dashboard como se ha realizado anteriormente.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Seleccione la opción **Manual input**, añade el comportamiento para cambiar el valor numérico cuando se pulsa el botón, esto hará que se envíe un mensaje al topic de MQTT correspondiente.

Para facilitar el uso del protocolo MQTT en el programa use la librería **PubSubClient** que habrá instalar mediante el gestor de paquetes. A continuación importe la libreria y cree el cliente MQTT y le pasamos el cliente WiFi.



Una vez creado el cliente, habrá que pasarle la configuración, que será dos variables una con el endpoint y la otra el puerto del bróker MQTT que en este caso usaremos Ubidots.

Texto

Descripción generada automáticamentePara realizarlo de una manera más sencilla, aquí está el código utilizado para realizar la conexión mediante MQTT a Ubidots.

Como se puede observar, necesitará tanto el token al igual que en el protocolo HTTP y necesitará la información sobre el topic al que se va a conectar, el cual será:



Texto

Descripción generada automáticamentePodrá ver como se realiza con éxito la conexión tanto a MQTT como la **suscripción al topic**, ejecuta el código y obtendrá lo siguiente.

Una vez finalizado lo anterior, ya solo quedaría **modificar el envio** de los datos anteriormente implementado por HTTP a una **nueva versión con MQTT** e implementar lógica en el código para que cuando llegue un mensaje por MQTT el dispositivo quede **suspendido**, esto quiere decir que no realice ninguna medición y que muestre por pantalla la palabra suspendido. Hasta que se reinicie o llegue otro mensaje.

Para este ultimo apartado de la parte 1 de la practica tendremos que eliminar todo lo anteriormente creado por HTTP y adaptarlo a MQTT.

Texto

Descripción generada automáticamenteTexto

Descripción generada automáticamentePara adaptar la lógica de nuestro código tendrá que modificar la función loop() la cual contiene desde el primer momento todo el código asociado al sensor de ritmo cardiaco y oxigeno de la practica 2.Implementa el siguiente código, posteriormente habrá una explicación de su funcionamiento.

Condición para saber si el dedo esta situado en el sensor, gracias a la función getIR() que recoge los valores del led infrarrojo del sensor

Función que obtiene las pulsaciones detectadas por el sensor.

Texto

Descripción generada automáticamente

Si está suspendido ya que ha llegado un mensaje MQTT lo muestra

Esta parte del código muestra tanto en el serial como en nuestra pantalla las pulsaciones obtenidas por el sensor.

Todo este código como se puede observar estará contenido en un gran bloque if, else; el cual controlará los mensajes recibidos por MQTT, ya que si recibe un mensaje habrá que dejar el sensor suspendido para que no analice hasta nuevo aviso ( otro mensaje MQTT). En este caso para hacer la implementación mas sencilla de entender implementa una función auxiliar para controlar esta variable suspended.

Antes de entrar en detalle en la parte de suspender al recibir un mensaje MQTT, vamos a ver como se envía la información a Ubidots mediante MQTT. Para ello haz uso de una función auxiliar que se encargue exclusivamente de ello.

Texto

Descripción generada automáticamente

Una vez entendido este apartado, implementa una nueva función auxiliar que será la llamada en el bucle principal al hacer mqttclient.loop()

Implementa la lógica pedida en la practica para que cuando llegue un mensaje por MQTT el dispositivo quede **suspendido**, esto quiere decir que no realice ninguna medición y que muestre por pantalla la palabra suspendido. Hasta que se reinicie o llegue otro mensaje.

Texto

Descripción generada automáticamente

Como puede observar, parsea el mensaje recibido del topic suscrito y si es distinto el numero recibido al que se había recibido anteriormente entonces la variable suspendido valdrá 1 y el loop() principal no hará nada, no tomará mas mediciones hasta que el mismo numero vuelva a ser recibido por MQTT.

Con esto se finaliza el bloque 1 de la practica que estaba dirigido a la comunicación mediante distintos protocolos.

**Parte 2**

En este apartado de la práctica se realizarán pruebas con las funcionalidades de tareas disponibles en el sistema operativo incluido en los MCU ESP32. Introducir el dispositivo en primera instancia hará que sea más sencillo su utilización y programación.

Creado y desarrollado por Espressif Systems, ESP32, una serie de microcontroladores de bajo costo y de bajo consumo con sistema en chip con WiFi y Bluetooth de modo dual integrados. El procesador tiene dos núcleos de procesamiento cuyas frecuencias operativas pueden controlarse independientemente entre 80 megahercios (MHz) y 240 MHz.

Imagen de la pantalla de un computador

Descripción generada automáticamente con confianza baja

Una vez tenga una idea del microcontrolador que utilizará, proceda a vincularlo con Arduino IDE. Para ello en primer lugar tendrá que instalar ciertas librerías para su utilización. Estas serán:

* Arduino\_ESP32\_OTA

Una vez instalada, si se realiza la practica con Windows puede que el IDE no le detecte el **port**, por lo que para ello tendrá que instalar los drivers necesarios de la placa. Para ello, siga los siguientes pasos:

En primer lugar para confirmar que el driver no esta instalado y que el problema es este, vaya a **Administrador de dispositivos** en su sistema Windows.

Cuando no se encuentra instalado el driver en la PC, por lo regular, la tarjeta ESP32 aparece en Otros dispositivos como CP2102 USB to UART Bridge Controller, con un símbolo de warning ⚠️.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Si ya comprobamos que realmente no está instalado el driver, descargamos el instalador desde la página oficial de SILICON LABS, para Windows mediante el siguiente enlace:

<https://www.silabs.com/developers/usb-to-uart-bridge-vcp-drivers?tab=downloads>

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamenteComo recomendación, instale el que se indica seguidamente, ya que es el que viene con instalador.

Posteriormente, descomprima y ejecute el archivo que corresponda a su PC. Si es de 64 bits, ejecute el archivo CP210xVCPInstaller\_x64.exe

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamenteCuando finalice la instalación, reinicie el PC para que se apliquen los cambios, una vez realizado vuelva a Arduino IDE, y ya verá como le detecta el microcontrolador.

Una de las principales ventajas del ESP32 sobre las placas Arduino tradicionales es su capacidad para realizar tareas en paralelo gracias a sus dos núcleos de procesamiento Xtensa LX6 de 32 bits: core 0 y core 1. Mientras que la mayoría de las placas Arduino están basadas en microcontroladores de un solo núcleo, el ESP32 tiene dos núcleos de procesamiento que pueden funcionar de forma independiente uno del otro. Una vez terminada esta breve introducción y configuración del dispositivo proceda con la práctica.

Texto

Descripción generada automáticamente

En esta primera parte se creará un **script con dos tareas una que imprime un mensaje y la otra que enciende y apaga el led integrado en la placa**.

Para realizar esto de manera sencilla vamos a ver unos concetos sobre la programación de la placa.

* xPortGetCoreID() sirve para identificar que núcleo ejecuta la tarea

Arduino IDE es compatible con FreeRTOS para ESP32, que es un sistema operativo en tiempo real. Esto permite manejar varias tareas en paralelo que se ejecutan de forma independiente. Las tareas son fragmentos de código que ejecutan algo. Por ejemplo, puede hacer parpadear un LED, realizar una solicitud de red, medir lecturas de sensores, publicar lecturas de sensores, etc.

* TaskHandle\_t Task1 sirve para crear un identificador de una tarea

Texto

Descripción generada automáticamentePosteriormente para configurar la tarea creada, tendrá que utilizar la siguiente función:

Una vez ya creada la tarea, podrá implementar la función que necesita ser ejecutada por esa tarea.

Captura de pantalla de un celular

Descripción generada automáticamenteComo se puede intuir, si es capaz de crear una tarea, del mismo modo será capaz de eliminarla, para ello dispone de la siguiente función:

* vTaskDelete(Task1); elimina la tarea pasa por parámetro

Para implementar el primer apartado de la segunda parte simplemente tendrá que hacer esto dos veces y realizar en un núcleo la impresión del mensaje y en el otro llevar el control de la led integrada. Podemos verlo reflejado de la siguiente manera:

Interfaz de usuario gráfica, Diagrama, Texto

Descripción generada automáticamente con confianza media

**Task1Code()**

Print name of task, and core

**Task2Code()**

Blink led every 5sec

Texto

Descripción generada automáticamentePara realizar lo anterior, implemente dos funciones y dos creaciones de tasks en la función setup(), de la siguiente manera.

Y simplemente necesitará las funciones de control de cada tarea.

Texto

Descripción generada automáticamente

Como puede observar una vez ejecutado una led azul se enchufará cada 5 segundos en el board y cada segundo se imprimirá en el Serial el mensaje siguiente:

Con esto se da por concluida la implementación del primer apartado, pero faltaría ver lo siguiente, **¿Pese al DELAY, las tareas se ejecutan en paralelo?**

El sistema de temporización de FreeRTOS se basa en "ticks", que son unidades de tiempo básicas que son controladas por un "tick interrupt". Este "tick interrupt" ocurre a intervalos regulares y es responsable de aumentar un contador de interrupciones interno, mantener la temporización de tareas y administrar la planificación del sistema.

La función vTaskDelay() suspende temporalmente y se pone en estado de bloqueo durante un número determinado de ticks cuando una tarea la llama. La tarea no se ejecutará durante este tiempo y estará en espera hasta que pase el número de ticks especificado.

Por lo tanto, otras tareas del sistema que están listas para hacerlo seguirán ejecutándose incluso si una tarea esté bloqueada debido a un vTaskDelay(). Esto permite que En sistemas que utilizan FreeRTOS, como ESP32, las tareas se pueden ejecutar en paralelo.

Siguiendo con lo mencionado, ahora tendrá que adaptar el código a la siguiente funcionabilidad: Conecta el emisor led de la práctica 2 al esp32 de forma similar a como hiciste con Arduino. Programa ahora **tres tareas, una por cada color del RGB**, estas tareas deben encender y apagar el color correspondiente siguiendo los siguientes tiempos: 1000 ms para el rojo. 2000ms para el azul. 5000 ms para el verde.

Como se mencionó con anterioridad, para esta pequeña modificación simplemente habrá que añadir una nueva tarea y en las funciones correspondientes a cada una de ellas plantear la lógica de enchufado y apagado de los leds.

Texto

Descripción generada automáticamente

Como se puede observar el pin rojo se enchufa cada segundo, el azul cada 2 y el verde cada 5. Esto ocurre al igual que en el anterior apartado gracias a la función:

vTaskDelay(pdMS\_TO\_TICKS(X000)); //Siendo X el valor que necesitemos

Para implementar la lógica del enchufado y apagado de los leds simplemente implementa las siguientes funciones.

Texto

Descripción generada automáticamentePara el led rojo:

Texto

Descripción generada automáticamentePara el led azul:

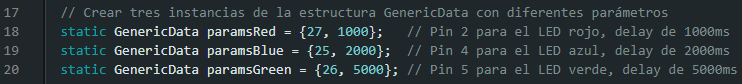
Texto

Descripción generada automáticamentePara el led verde:

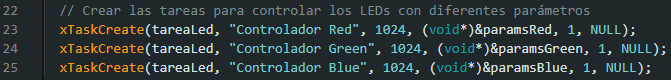
Con esto puedes observar como el modulo led RGB va cambiando de color, a veces y como consecuencia de que estén trabajando las tareas en paralelo, podrán solaparse los colores, entonces obtendrás colores como el blanco, el morado o el naranja.

Para finalizar esta segunda parte de la práctica, realiza otra modificación para que estas funciones de **control sobre led estén implementadas en una sola tarea**. Para implementar esto realiza una única función a la cual llamarás tres veces mediante la misma tarea con distintos parámetros.

Primero y antes que nada nos crearemos la estructura del tipo de dato y además crea la tarea, que se encargue de llamar tres veces a la función con los distintos tipos de led.

A continuación, inicializa los leds, de la siguiente manera.

Una vez realizo esto, simplemente crea las tareas ( llamadas a la función que se encargará de enchufar o apagar los led ).



Y por último para la gestión de leds, crea esta función para su procesamiento:

Texto

Descripción generada automáticamenteCon esto como en la anterior parte, veras distintos colores ya que hay solapes en la ejecución de las tareas.

**Parte 3**

En esta última parte simplemente habrá que familiarizarse con el modo bajo consumo del ESP32, que reduce considerablemente su consumo y sus prestaciones, esto es utilizado para aumentar la duración de la batería entre medidas

En esta primera parte se pide sobre el código de la parte 2, añadir la librería de sueño y crear un script que realice diferentes funcionalidades.

En primer lugar tendrá que realizar un script que implemente lo siguiente: Encender un led verde cada 100ms por 10 veces y una vez realizado se encienda un led rojo y se duerma por 15 segundos.

Texto

Descripción generada automáticamente

Como se puede apreciar en el código, implementar primero las diez ejecuciones de la led Verde para posteriormente poner el pin de la led rojo en HIGH y ponerlo a dormir por 15 segundos.

En este caso, la función **gpio\_hold\_en() se encargará de mantener encendido el led rojo** cumplimentando asi la pregunta 3, configura una alarma de temporizador para despertar después de 15 segundos, mantiene el estado del pin rojo durante el sueño profundo y entra en el modo de sueño profundo, donde el sistema se detiene hasta que se activa la alarma del temporizador.

Se pide **añadir un contador de reinicios** para cada vez que se ponga a dormir, para ello añade la siguiente línea:

Para **implementar la pregunta 4 y 5**, simplemente habrá que añadir dos nuevos pines de nuestro microcontrolador, los cuales se encargaran de cortocircuitar la ejecución, mediante la conexión de un jamper cuando se este ejecutando el código. En este caso, habrá que añadir, lo siguiente al código de la anterior pregunta de la parte 3:

Mano de una persona

Descripción generada automáticamente con confianza baja

Utilizaremos el pin numero 4 como cortocircuito y el pin numero 14, como pin que en realidad no hace nada.

Como se puede apreciar uno de los jumpers conectados al Lolin 32, solo se encuentra conectado por uno de sus extremos, este permitirá interrumpir el modo sueño y poner a trabajar otra vez al microcontrolador, con esto se puede entender mejor los modos de eficiencia y bajo consumo de los mismos, ya que estos pueden estar enchufados pero esperando alguna causa que active al mimso y desenboque en un efecto. Para implementar esto último en el codigo simplemente modifica lo anterior de la siguiente manera.

Texto

Descripción generada automáticamente

Como se puede apreciar el pin que no se utiliza simplemente esta en HIGH todo el rato, pondrá el microcontrolador a dormir pero siempre dejando los pines rojos y el unused en HIGH para que sigan activos mediante la función nombrada anteriormente, por último, la función esp\_sleep\_enable\_ext0\_wakeup(), permitirá despertar el microcontrolador en caso de que algo externo suceda, en este caso y como se especifica en los parámetros, si el pin wakeUpPin se pone a HIGH.

Como último apartado de la practica se intenta **implementar un código que especifique cuanto tiempo ha pasado en el modo deepsleep el MCU** y que especifique porque ha sido despertado mediante la función esp\_sleep\_get\_wakeup\_cause() y ESP32Time.h

Texto

Descripción generada automáticamente

Para ello implementa lo siguiente:

Con lo anterior implementado, obtendrá como salida algo parecido a lo siguiente:

El microcontrolador estuvo dormido durante: 18446744073709428813 microsegundos

El microcontrolador fue despertado por una causa desconocida

Estoy haciendo trabajo

20

Estoy haciendo trabajo

20

Indicando el tiempo como se pedía en el enunciado de la practica.