Sistemas Embebidos

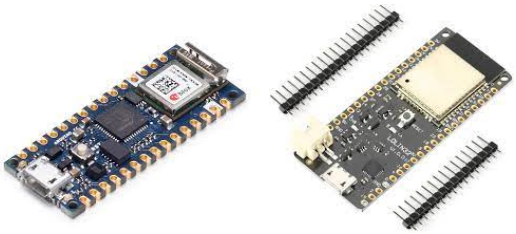
Práctica 3: Conectividad, tiempo real e interrupciones en dispositivos embebidos

Objetivos

- Prueba de tecnologías de comunicación del dispositivo embebido.
- Instalación y configuración de dispositivo en conexión con una plataforma en la Nube.
- Adquirir conocimientos y habilidades de envío y recepción de mensajes con el protocolo MQTT. Adquirir conocimientos de programación en tiempo real y uso de interrupciones

Descripción

En esta práctica haremos uso de dos microcontroladores diferentes para realizar tareas de comunicación y probar diferentes funcionalidades. Utilizaremos el microcontrolador Arduino Nano IoT 33 y el microcontrolador Esp32 en su versión board Lolin32



El MCU **Arduino IoT** es un dispositivo sencillo para construir aplicaciones de internet de las cosas. Con este MCU ya hemos realizado nuestro primer dispositivo embebido con el que obtenemos los datos de un sensor. El objetivo de esta práctica es utilizar la capacidad de conectividad de este MCU para añadir conectividad a nuestro dispositivo.

Alternativamente, trabajaremos con ESP32 para programar tareas en tiempo real y aprender sobre el uso de interrupciones y los diferentes modos de bajo consumo presentes en el microprocesador.

Trabajo a realizar:

Parte 1: Comunicación

Para la comunicación utilizaremos una plataforma existente llamada Ubidots Steam, esta plataforma nos permite crear paneles de datos donde mostrar la información de los dispositivos IoT.

1. Configurar Ubidots Steam
    a. Crea una cuenta en el portal de Ubidots stem
    b. Crea un dispositivo desde el menú superior llamado Nano-Iot-Test, utiliza el formato dispositivo en blanco.

c. Dentro del dispositivo crea una variable denominada “pulsaciones”. Dentro del rango permitido establece el rango desde 0 a 220
d. Dentro del cuadro de mando creado por defecto realiza las siguientes modificaciones:
    i. Elimina los datos de prueba.
    ii. Añade un widget de imagen con una foto de tu dispositivo.
    iii. Añade un widget de métrica que muestre el último valor de la variable de pulsaciones.
    iv. Añade un widget de line chart que represente los últimos valores de la variable pulsaciones.
e. Obtén las credenciales necesarias para enviar los datos a ubidots, necesitarás el label del device y el label de la variable. Obtén tu API key desde el menú de usuario - Api credentials.
f. Establece la conexión a la Wifi de tu dispositivo móvil o Wifi de casa. En la Universidad es probable que necesites generar una red wifi para poder conectar el dispositivo.
g. Crea un código para el dispositivo que se conecte a la red wifi, utiliza la librería arduino-libraries/WiFiNINA, y usando las credenciales facilitadas por el portal envíe un entero aleatorio entre 60 y 120 que representa la media de las pulsaciones. Haz que esté mensaje se envíe cada 5 segundos con un valor distinto. Puedes utilizar la documentación de API para HTTP facilitada por la plataforma. https://docs.ubidots.com/v1.6/reference/welcome
h. Observa los cambios en el valor de la variable en el “Dashboard” a lo largo del tiempo.

2. Enviar y recibir datos por MQTT.
    a. Crea un botón en el portal de la plataforma y 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.
    b. Crea un código para el dispositivo que una vez conectada a la red wifi se conecte al servidor de MQTT de ubidots y haz que se suscriba al topic correspondiente para esperar los mensajes del servidor.

3. Reemplaza en tu prototipo el Arduino UNO usado en la **práctica 2** por el dispositivo Arduino nano IoT y verifica que las conexiones son correctas teniendo en cuenta el pinout del nuevo dispositivo.
    a. Modifica el código utilizado para la práctica 2 para que se conecte a una red wifi. Realiza también las modificaciones necesarias para que el dispositivo envíe cada 5 segundos los datos del sensor a la plataforma mediante MQTT.
    b. Modifica 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.

Parte 2: Tiempo real y FreeRTOS

En este apartado de la práctica vamos a realizar pruebas con las funcionalidades de tareas disponibles en el sistema operativo incluido en los MCU ESP32.

Prepara un proyecto para poder programar con el MCU ESP32. Tendrás que utilizar como board WEMOS Lolin32.

1. Crea un primer script que cree dos tareas para queso ejecuten en paralelo. En la primera de ellas escribe el código para que cada 100 ms se escriba por serial el mensaje: *Hola soy la tarea "Nombre task", me estoy ejecutando en el core "Número core".* La segunda tarea debe encender y apagar el built in led del dispositivo, puedes hacerlo con el pin *BUILTIN_LED*, cada 5 segundos. Observa el comportamiento del script. ¿Pese al *DELAY*, las tareas se ejecutan en paralelo?.
2. 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.
3. Modifica tu código anterior para que solo exista una tarea que se lanza tres veces y utiliza los parámetros de la tarea para enviar el tiempo de *DELAY* y el pin que debe encender.

Parte 3: Interrupciones

El MCU ESP32 viene preparado con un modo de bajo consumo comúnmente llamado *deepsleep*. En este modo el MCU reduce considerablemente su consumo y sus prestaciones, esto es utilizado para aumentar la duración de la batería entre medidas.

Para poder utilizar el modo *deepsleep* necesitamos incluir la librería *esp_sleep.h,* la primera forma de dormir el procesador es especificando cuantos segundos debe dormir.

1. Crea un nuevo proyecto para esta parte de la práctica con la misma configuración que el apartado anterior. Añade la librería de sueño y crea un script que realice lo siguiente:
    a. Enciende el led verde y muestra por serial el mensaje "Estoy haciendo trabajo" cada 100 ms un total de 10 veces.
    b. Enciende el led en color rojo.
    c. Entra en modo sueño durante 15 segundos.

    ¿Qué ocurre cuando el MCU se duerme? ¿Y cuándo se despierta?
2. Modifica el script anterior para que se guarde el número de reinicios que se han realizado y muestra esté número por serial. Asegúrate de que solo se cuentan los reinicios propios del modo sueño. *esp_wake_deep_sleep()*
3. Busca la forma de mantener el LED encendido incluso cuando el MCU esté dormido y observa que el led esté en verde si el MCU está activo y en rojo cuando esté en modo sueño.
4. Modifica tu código anterior para poner en modo *HIGH* un pin válido que no estés utilizando, configúralo para que el valor se mantenga en alto incluso en el modo sueño. Conecta un Cable *jumper* al pin y deja el otro extremo sin conectar.
5. Configura el pin 4 para que despierte al MCU si se detecta un valor *HIGH.* Utilizando el cable, toca el pin 4 para despertar el microcontrolador. Prepara ahora una función en tu código que especifique cuanto tiempo ha pasado en el

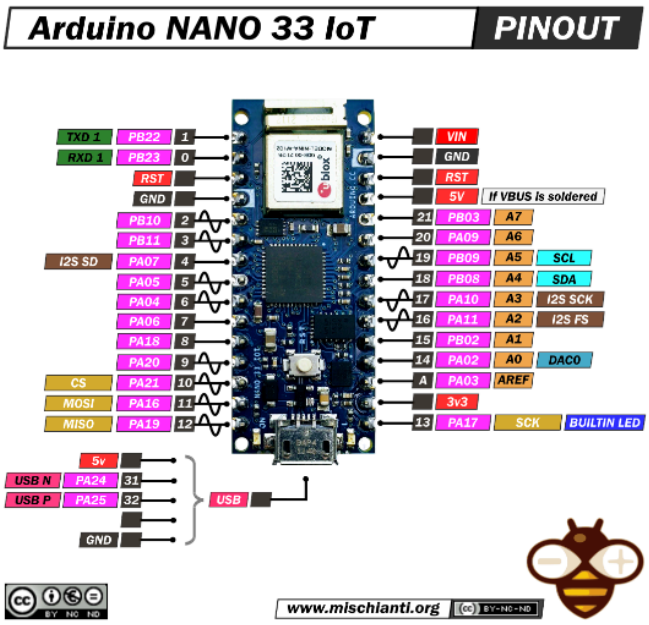
modo *deepsleep* el MCU y que especifique porque ha sido despertado.
esp_sleep_get_wakeup_cause() y ESP32Time.h

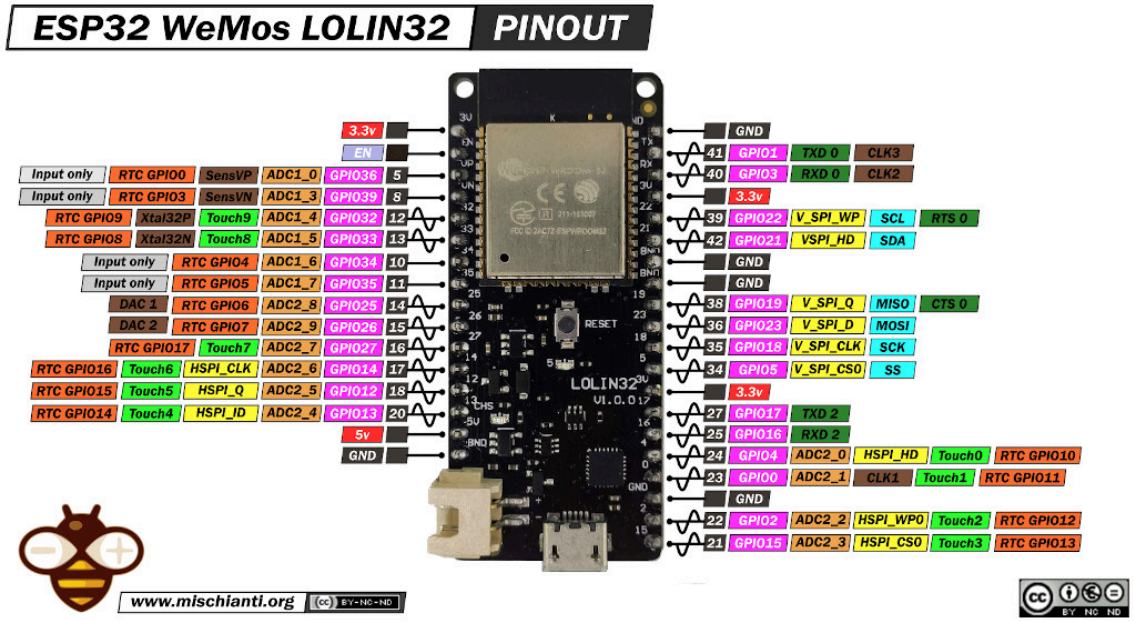
Ponte en contacto con el profesor de la asignatura si no dispones de los materiales para realizar la práctica.

Aclaraciones para la práctica:

A la hora de integrar todos los componentes, en la anterior práctica utilizamos el pin de 5v de Arduino para alimentar los sensores y pantalla. Cuando tengas que hacer esté paso para el Arduino IoT utiliza el pin de 3.3v, ya que en la placa el pin de 5v está desactivado por defecto y hay que activarlo con una soldadura que no hemos realizado.

Si en algún momento no llegas a leer los mensajes del serial que ocurren durante el setup() añade el siguiente código después de Serial.begin(9600); while(!Serial); de esta forma el programa esperará a que Serial esté disponible para ser usado antes de continuar.





Bibliografía:

Considera consultar los siguientes materiales.

https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/system/sleep_modes.html

https://docs.espressif.com/projects/esp-idf/en/v4.3/esp32/api-reference/system/freertos.html

https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/peripherals/gpio.html

Normas de entrega:

- La realización del trabajo es individual.
- La fecha límite de entrega es el **21 de abril de 2024**.
- La entrega se realizará a través de la herramienta de entrega de trabajos de Campus Virtual.
- Los formatos válidos del documento son *MS Word* (.doc, .docx), *OpenDocument* (.odt) o *Portable Document Format* (.pdf). Entrega en un fichero zip el documento junto a los recursos que consideres oportunos.