PRÁCTICA ESP32

1. Introducción

Este informe describe el diseño, desarrollo y funcionamiento de un sistema embebido basado en el microcontrolador ESP32, cuyo propósito es medir distancias mediante un sensor ultrasónico, calcular una aceleración simulada en función de dicha distancia y mostrar los resultados tanto de forma visual a través de cuatro LEDs, como de manera remota mediante un servidor web local.

El proyecto combina la lectura de sensores, la comunicación inalámbrica WiFi, la gestión de modos de bajo consumo y el almacenamiento persistente de datos en memoria EEPROM.

Esta práctica constituye una primera fase del proyecto final que tiene como objetivo la creación de un minicoche autónomo, capaz de ajustar su aceleración o frenado según la distancia al vehículo u obstáculo que tenga delante.

2. Descripción del funcionamiento

El núcleo del sistema está compuesto por el microcontrolador ESP32 y el sensor ultrasónico HC-SR04, encargado de medir la distancia entre el sensor y el objeto situado frente a él. Los pines de control del sensor se definen al inicio del código con las sentencias:

```
#define TRIG_PIN 14
#define ECHO_PIN 12
```

El pin TRIG emite un pulso ultrasónico de corta duración, mientras que el pin ECHO mide el tiempo que tarda dicho pulso en reflejarse y volver. En el código, esto se implementa dentro de la función medirDistancia(), que utiliza digitalWrite() para generar el pulso de disparo y la función pulseIn() para medir el tiempo de respuesta. El cálculo de la distancia en centímetros se realiza con la fórmula:

```
float distancia = duracion * 0.034 / 2;
```

Donde 0.034 representa la velocidad del sonido en centímetros por microsegundo y la división entre dos corresponde al viaje de ida y vuelta de la onda. En un futuro se plantea la inclusión de un sensor de temperatura ya que la velocidad de la onda depende de esta.

Una vez obtenida la distancia, el programa ejecuta una función que simula una aceleración proporcional a la distancia medida. Para ello, se define un rango entre 15 cm y 100 cm, en el que a la distancia mínima corresponde la aceleración más baja y a la máxima la aceleración más alta. El valor se calcula mediante una interpolación lineal, asegurando una transición progresiva y coherente con el comportamiento esperado de un vehículo que acelera conforme aumenta la distancia de seguridad.

Los resultados se representan mediante cuatro LEDs conectados a los pines 19, 18, 5 y 17 del ESP32. La función mostrarLEDS() gestiona su encendido en función del valor de aceleración. Cuando el valor es mínimo, solo se enciende el primer LED, y a medida que la aceleración simulada aumenta, se activan los siguientes. De esta manera, los LEDs actúan como una visualización en tiempo real de la magnitud de la aceleración calculada.

Además, la distancia mínima registrada desde el inicio del programa se almacena de forma persistente en la EEPROM. Esto se consigue mediante la biblioteca <EEPROM.h> y el uso de las funciones EEPROM.readFloat() y EEPROM.writeFloat(). Cada vez que el sistema detecta una nueva distancia inferior a la guardada, se actualiza el valor en memoria y se confirma por el puerto serie y el servidor web. De esta forma, aunque el dispositivo se reinicie, la distancia mínima se conserva, garantizando la integridad del dato.

3. Comunicación WiFi y servidor web local

La comunicación inalámbrica se implementa mediante la biblioteca WiFi.h. Al inicio del programa se definen las credenciales de conexión:

```
const char* ssid = "yiyiyi";
const char* password = "xabicrack";
```

En la función setup(), el ESP32 se conecta a la red con WiFi.begin (ssid, password) y entra en un bucle de espera hasta obtener una conexión exitosa. Una vez establecida, la IP local se imprime por el puerto serie con WiFi.localIP(), lo que permite acceder posteriormente al servidor desde un navegador. Esta IP es 192.168.57.25.

El servidor web se crea utilizando la clase WiFiServer, que escucha peticiones HTTP entrantes en el puerto 80. Cada vez que un cliente, en este caso un móvil, accede a la dirección IP del ESP32, el programa genera una página HTML desde la función handleRoot(). Esta página muestra tres datos principales: la distancia actual, la distancia mínima registrada y la aceleración simulada.

El contenido HTML se genera directamente dentro del código utilizando concatenación de cadenas, y los valores de las variables se insertan dinámicamente en cada actualización. Por ejemplo:

```
String html = "<html><body>";
html += "<h1>Monitor de distancia</h1>";
html += "<p>Distancia actual: " + String(distanciaActual) + " cm</p>";
html += "<p>Distancia mínima: " + String(distanciaMinima) + " cm</p>";
html += "<p>Aceleración: " + String(aceleracion) + " cm/s²</p>";
html += "</body></html>";
```

De este modo, la página muestra siempre la información actualizada sin necesidad de hardware adicional. Esta interfaz web resulta muy útil para la depuración y la visualización remota, ya que permite supervisar en tiempo real el comportamiento del sistema desde cualquier dispositivo conectado a la misma red WiFi.

4. Modo de bajo consumo (Light Sleep)

El sistema incorpora un modo de ahorro energético que se activa cuando el vehículo se encuentra demasiado cerca del obstáculo (menos de 15 cm durante más de 5 segundos) o cuando el usuario pulsa un botón conectado al pin GPIO26.

Para mejorar la fiabilidad, la activación por pulsador ahora utiliza una interrupción de hardware con debounce digital implementado en software. La ISR (Interrupt Service Routine) solo marca que hubo pulsación (flagISR = true), y en el bucle principal se verifica esta marca aplicando un retraso de debounce de 1000 ms con millis(). Este enfoque evita los rebotes del botón y garantiza que el ESP32 solo entre en Light Sleep una vez por pulsación.

Cuando se cumplen las condiciones de sleep, se ejecuta la función ejecutarSleep(), que imprime por el puerto serie: “Entrando en modo baja energía (light sleep)...”

Luego apaga los LEDs, configura el pin de wake-up con esp_sleep_enable_ext0_wakeup() y entra en esp_light_sleep_start(). Al despertar, imprime: “Despertado por pulsador”

El modo Light Sleep se seleccionó por su equilibrio entre consumo reducido y rápida reactivación. A diferencia del modo Deep Sleep, este conserva parte del estado interno del procesador y de las variables, permitiendo una reanudación casi inmediata tras la interrupción externa generada por el pulsador o el evento de proximidad.

5. Almacenamiento en EEPROM

La EEPROM cumple la función de conservar la distancia mínima detectada a lo largo del tiempo. En la fase de inicialización, el programa lee el valor guardado mediante EEPROM.readFloat(0) y lo carga en la variable distanciaMinima. Durante la ejecución, si la función medirDistancia() obtiene un valor menor que el actual, se actualiza el registro en memoria mediante EEPROM.writeFloat(0, distanciaMinima) seguido de EEPROM.commit().

De esta manera, la EEPROM actúa como una memoria no volátil que permite conservar datos entre reinicios, garantizando la continuidad de la información crítica para el sistema. En este caso se puede considerar que si la distancia mínima guardada es 0 el coche ha tenido un choque.

6. Problemas encontrados

A lo largo del desarrollo del proyecto han surgido diversos inconvenientes tanto de software como de hardware. Uno de los principales fue la inestabilidad del sensor ultrasónico, que en ocasiones devolvía lecturas erráticas debido a rebotes de señal o a la baja calidad de la reflexión del ultrasonido. Este problema se ha mitigado ajustando los retardos entre pulsos (delayMicroseconds(10)) y limitando el tiempo de espera de pulseIn() para evitar bloqueos prolongados.

También se presentaron dificultades al configurar los pines de interrupción RTC para el modo de bajo consumo, ya que no todos los GPIO del ESP32 son compatibles con funciones de wake-up. Tras revisar la documentación técnica del microcontrolador, se ha determinado que el GPIO26 era una opción válida y se ha adaptado el código en consecuencia.

Otro problema recurrente fue la actualización incorrecta de la página web durante la comunicación WiFi. En las primeras versiones del programa, los valores mostrados no se refrescaban adecuadamente debido a que las variables globales no se actualizaban antes de servir la respuesta HTTP. Esto se ha solucionado asegurando que la función handleRoot() accediera a los valores más recientes de las variables distanciaActual, distanciaMinima y aceleracion justo antes de construir el HTML.

Por último, se ha detectado un comportamiento anómalo en los LEDs tras el despertar del modo de bajo consumo. En algunos casos, los pines quedaban en un estado indeterminado. Para solucionarlo, se añadió un bloque de reinicialización en la función setup() y un apagado controlado dentro de entrarEnSleep().

7. Conclusión y proyección hacia el proyecto final

Se ha logrado implementar correctamente la medición de distancia mediante ultrasonidos, el cálculo dinámico de una aceleración simulada que posteriormente puede ser mejorada, la representación visual con LEDs, la comunicación WiFi a través de un servidor web y el almacenamiento persistente en EEPROM.

El uso del modo Light Sleep permite un equilibrio entre consumo energético y capacidad de respuesta, mientras que la interfaz web posibilita la monitorización remota en tiempo real. Además, la correcta gestión de los estados de los LEDs y la persistencia de los datos fortalecen la fiabilidad del sistema.

Durante el desarrollo se enfrentaron distintos problemas de hardware y software que exigieron comprender en profundidad el funcionamiento interno del ESP32, del sensor ultrasónico y de la biblioteca WiFi. La resolución de estos desafíos contribuyó a afianzar los conocimientos sobre depuración, sincronización y eficiencia energética en sistemas embebidos.

Este proyecto constituye la base para el proyecto final, que consistirá en un minicoche autónomo controlado por un ESP32 capaz de modificar su aceleración y frenado en función de la distancia con respecto al vehículo u objeto delantero. En esa versión ampliada, la aceleración simulada calculada en esta práctica se convertirá en una señal de control real para los motores del vehículo, aplicando los mismos principios de seguridad y respuesta adaptativa.