Universidad de Guadalajara

Piensa y trabaja

REPORTE DE PRÁCTICA 5

Práctica (Lectura de un potenciómetro y control de salida PWM)

Participantes:

Lorena Villalobos Carrillo

Ximena Lucía Rodríguez Oliva

Brayam De Jesus De La Cerda Valdivia

Seminario de problemas de programación embebidos

(D01 –I9893)

Fecha: 27/02/2025

Universidad de Guadalajara

Piensa y trabaja

1. Resumen

En esta práctica se implementó un sistema utilizando una **Tarjeta de Desarrollo ESP32** para leer el valor de un **potenciómetro** y generar una salida **PWM** proporcional. Se empleó un conversor **ADC** para obtener la lectura del voltaje del potenciómetro y, posteriormente, se procesó para obtener un ángulo en grados y un porcentaje de ciclo de trabajo. La salida PWM se aplicó a un pin de la ESP32 para observar su variación.

2. Objetivos

- Leer el valor analógico de un potenciómetro utilizando una Tarjeta de Desarrollo ESP32.
- Convertir la señal analógica en una representación útil (voltaje, ángulo y porcentaje).
- Generar una señal PWM basada en la lectura del potenciómetro.
- Comprender el manejo de entradas analógicas y salidas digitales en sistemas embebidos.

3. Introducción

Los sistemas embebidos permiten la interacción con el mundo físico mediante sensores y actuadores. En esta práctica, se utilizó un potenciómetro como entrada analógica y una **Tarjeta de Desarrollo ESP32** para procesar la señal y generar una salida **PWM** proporcional.
El **potenciómetro** es un divisor de voltaje que permite variar su resistencia manualmente, proporcionando un voltaje de salida dependiente de la posición de su perilla. La ESP32, mediante su conversor **ADC (Analog-to-Digital Converter)**, convierte esta señal en un valor digital que puede procesarse en software para generar distintos tipos de salidas.
La **modulación por ancho de pulso (PWM)** es una técnica ampliamente utilizada en sistemas embebidos para controlar la potencia entregada a dispositivos como motores, LEDs y otros actuadores. En esta práctica, se utilizó PWM para representar la salida del sistema en respuesta a la variación del potenciómetro.

4. Marco Teórico

4.1 Tarjeta de desarrollo ESP32

La **ESP32** es un microcontrolador de alto rendimiento que incluye conectividad Wi-Fi y Bluetooth, así como múltiples entradas analógicas y salidas digitales. Su capacidad de procesamiento y versatilidad lo hacen ideal para aplicaciones de control y automatización.

4.2 Potenciómetro

Un **potenciómetro** es un resistor ajustable que permite obtener un voltaje variable en función de la posición de su perilla. Se emplea comúnmente como interfaz para ajustar parámetros en circuitos electrónicos.

4.3 Conversión ADC

El **convertidor analógico-digital (ADC)** de la ESP32 permite transformar señales analógicas en valores digitales. En esta práctica, se utilizó para obtener la lectura del potenciómetro, la cual se procesó y escaló en diferentes representaciones.

4.4 Modulación por Ancho de Pulso (PWM)

La **modulación por ancho de pulso (PWM)** es una técnica que permite controlar la potencia entregada a un dispositivo variando el ciclo de trabajo de una señal digital. Es ampliamente utilizada para el control de motores, regulación de brillo en LEDs y conversión de señales analógicas simuladas.

5. Metodología

5.1 Materiales y Herramientas

- **Tarjeta de Desarrollo ESP32**
- **Potenciómetro**
- **Protoboard y cables de conexión**
- **Computadora con Arduino IDE**

{ñ{ñ{ñ

5.2 Explicación del Código

El código está estructurado en varias partes clave:

- **Inicialización y Configuración:** Se definen los pines del ESP32 asociados a los segmentos y dígitos del display.

```
const int segmentos[] = {14, 4, 5, 19, 18, 13, 21};
const int digitos[] = {23, 22, 25, 26};
```

Estas variables se declaran como const porque su contenido no cambia durante la ejecución del programa. Esto optimiza el uso de memoria y evita modificaciones accidentales en el código.

- **Gestor de Tiempo:**

```
while (millis() - lastMillis < w1s) {
    mostrarNumero(horas, minutos);
}
```

Este fragmento de código gestiona el tiempo de actualización de la pantalla. La función millis() devuelve el tiempo transcurrido desde el inicio del programa, asegurando que la función mostrarNumero() se ejecute durante un tiempo controlado antes de actualizar los minutos.

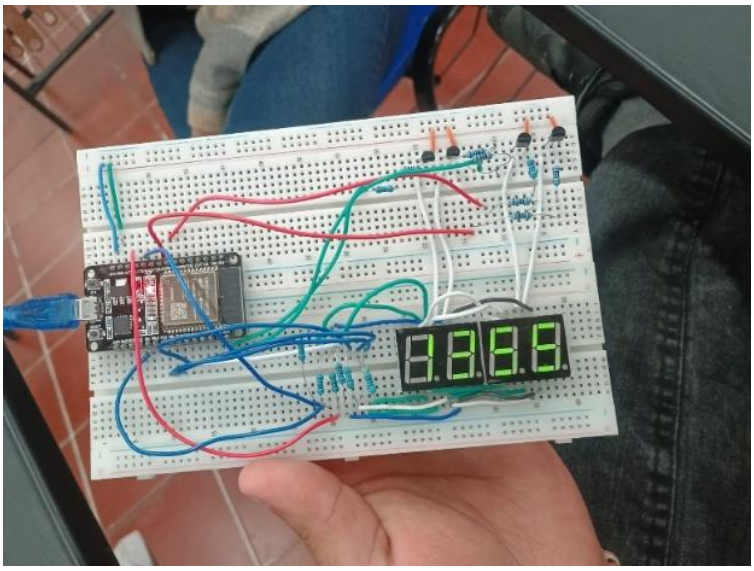
- **Multiplexación de los Dígitos:**

```
for (int j = 0; j < 15; j++) {
    for (int i = 0; i < 4; i++) {
        mostrar(cifras[i], digitos[i]);
        delay(2);
        apagar();
    }
}
```

Este fragmento maneja la visualización de los dígitos de forma secuencial:

- El primer for (j) asegura que cada dígito se muestre de manera repetida para evitar parpadeos.
- El segundo for (i) recorre los dígitos (horas y minutos) para mostrarlos secuencialmente.
- mostrar() enciende el dígito correspondiente, delay(2) permite su visualización y apagar() lo desactiva antes de mostrar el siguiente.

6. Resultados



7. Conclusión

Esta práctica permitió comprender el funcionamiento de los displays de 7 segmentos y su implementación en sistemas embebidos con ESP32. Se logró diseñar un contador de horas eficiente que muestra la información de manera clara y precisa en los displays, utilizando la técnica de multiplexación para optimizar el uso de los pines del microcontrolador.
La programación del ESP32 permitió implementar un control de tiempo preciso mediante la función millis(), asegurando que el sistema avanzara correctamente en el conteo de minutos y horas. Además, se optimizó la visualización evitando parpadeos no deseados a través del control secuencial de los dígitos.
Como dificultades, se presentaron algunos problemas iniciales con la configuración de los pines y la multiplexación, los cuales fueron corregidos con ajustes en el código. Este ejercicio proporcionó un conocimiento valioso sobre el control de periféricos y la gestión de tiempo en microcontroladores, sentando una base sólida para futuros proyectos más complejos en sistemas embebidos.