PRÁCTICA DE ARDUINO - CONTROL DE LED CON BOTÓN

NIVEL: 1
DURACIÓN: 2 horas
OBJETIVO: Los estudiantes aprenderán a controlar un LED mediante un botón utilizando una placa Arduino.
MATERIALES

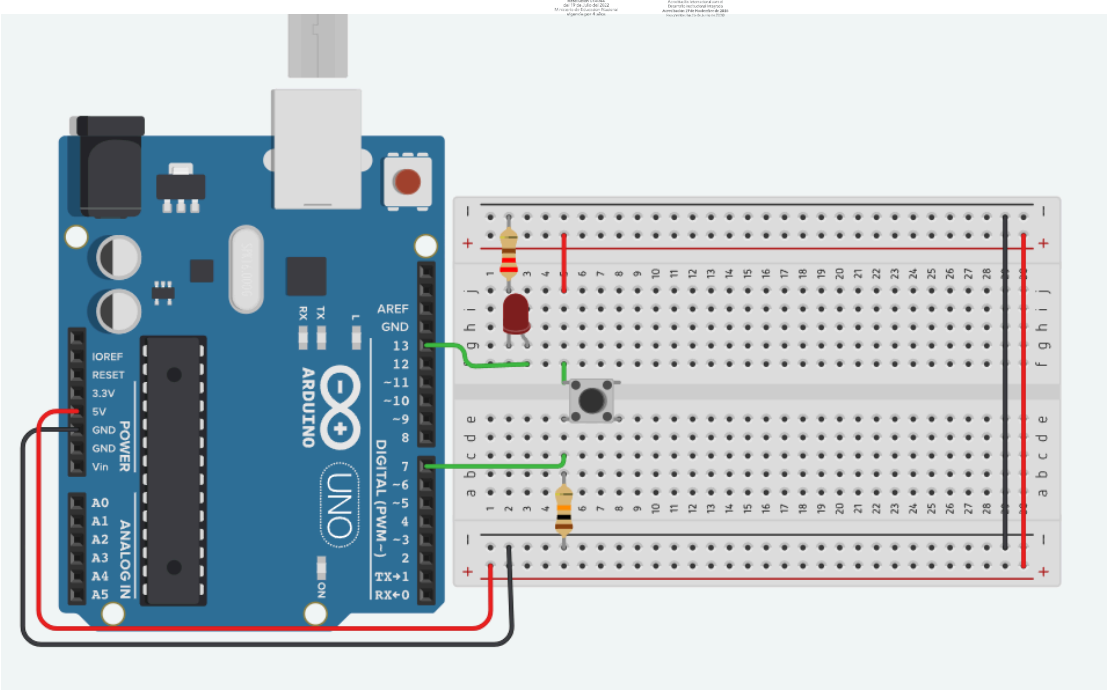
- Placa Arduino Uno (o similar)
- 1 LED
- 1 Resistencia de 220 ohmios
- 1 Resistencia de 10k ohmios
- 1 Botón
- Cables de conexión (jumpers)
- Protoboard
- Computadora con el software Arduino IDE instalado
- ThinkerCad

1. DESARROLLO DE LA PRÁCTICA

1.1 SIMULACIÓN EN THINKERCAD

1.1.1 COMPONENTES

1.1.2 MONTAJE



2.2 PROGRAMACIÓN EN ARDUINO IDE

```
// Definir los pines para el LED y el botón
const int ledPin = 13;      // Pin donde está conectado el LED
const int buttonPin = 7;    // Pin donde está conectado el botón
int buttonState = 0;        // Variable para almacenar el estado del botón

void setup() {
  // Configurar el pin del LED como salida
  pinMode(ledPin, OUTPUT);
  // Configurar el pin del botón como entrada
  pinMode(buttonPin, INPUT);
}

void loop() {
  // Leer el estado del botón
  buttonState = digitalRead(buttonPin);

  // Si el botón está presionado, encender el LED
  if (buttonState == HIGH) {
    digitalWrite(ledPin, HIGH);
  } else {
    digitalWrite(ledPin, LOW);
  }
}
```

pinMode(OUTPUT): Configura el LED como salida digital.
pinMode(INPUT): Configura el botón como entrada digital.
digitalRead(): Lee el estado del botón, si está presionado (HIGH) o no (LOW).
digitalWrite(): Controla el estado del LED, encendiéndolo o apagándolo.

2. ANÁLISIS DE RESULTADOS

- Pregunta 1: ¿Por qué necesitamos una resistencia en serie con el LED?

La resistencia en serie con el LED es necesaria para limitar la corriente que pasa a través del LED. Sin esta resistencia, el LED podría recibir demasiada corriente y quemarse. La resistencia ayuda a controlar la corriente a un nivel seguro para el LED y el Arduino. Generalmente, se utiliza una resistencia de 220 ohmios.

- Pregunta 2: ¿Qué función cumple la resistencia pull-down en el botón?

La resistencia pull-down evita que el pin digital del botón quede en un estado "flotante" cuando el botón no está siendo presionado. Un pin flotante puede generar lecturas aleatorias (ruido), causando un comportamiento inesperado en el sistema. La resistencia de 10k ohmios asegura que el pin del botón esté en un estado bajo (LOW) cuando no se presiona el botón, lo que garantiza que el Arduino lea correctamente el estado del botón.

- Pregunta 3: ¿Qué ocurre si no se incluye la resistencia pull-down?

Si no se incluye la resistencia pull-down, el pin digital del botón puede leer valores fluctuantes o incorrectos cuando el botón no está presionado. Esto ocurre porque el pin queda en un estado indefinido (flotante), lo que puede hacer que el LED se encienda o apague de manera aleatoria. Con la resistencia, garantizamos un estado estable cuando el botón no se está utilizando.

- Pregunta 4: ¿Qué sucede si se cambia el pin del LED en el código y no se modifica la conexión física?

Si en el código se cambia el pin del LED pero no se modifica la conexión física del circuito, el LED no funcionará como se espera. Por ejemplo, si el código se actualiza para usar el pin 12 en lugar del pin 13, pero el LED sigue conectado al pin 13 en la placa Arduino, el LED no se encenderá o apagará correctamente. Es crucial que el pin especificado en el código coincida con la conexión física en la placa.

- Pregunta 5: ¿Qué sucede si el botón se conecta incorrectamente en la protoboard?

Si el botón se conecta incorrectamente en la protoboard, no funcionará como se espera. En lugar de cambiar entre los estados HIGH y LOW cuando se presiona o suelta, el Arduino podría no detectar ningún cambio, o podría leer un estado incorrecto. Es importante asegurarse de que el botón esté conectado correctamente, con un lado al pin digital y el otro lado a GND a través de la resistencia pull-down.

- Pregunta 6: ¿Qué cambiarías en el código si quisieras que el LED permanezca encendido hasta que el botón se presione nuevamente (modo toggle)?

Para implementar un modo toggle (alternancia), donde el LED cambia de estado cada vez que se presiona el botón, se puede modificar el código de la siguiente manera:

```
const int ledPin = 13;
const int buttonPin = 7;
int buttonState = 0;
int lastButtonState = 0;  // Para almacenar el estado anterior del botón
int ledState = LOW;       // Para almacenar el estado actual del LED

void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin, INPUT);
}

void loop() {
  buttonState = digitalRead(buttonPin);

  // Si el botón ha sido presionado y está en estado HIGH
  if (buttonState == HIGH && lastButtonState == LOW) {
    ledState = !ledState;  // Cambia el estado del LED
    digitalWrite(ledPin, ledState);
  }

  // Almacena el estado actual del botón para la siguiente iteración
  lastButtonState = buttonState;
}
```

Explicación: El código anterior almacena el estado anterior del botón para detectar cuando el botón ha sido presionado y luego liberado. Esto permite cambiar el estado del LED (encendido/apagado) cada vez que se presiona el botón.

- Pregunta 7: ¿Por qué es importante el uso de pinMode() en el código?

El uso de pinMode() es crucial para indicar al Arduino cómo se comportarán los pines: si actuarán como entradas o salidas. En este proyecto:

a. El LED es una salida porque queremos enviar señales eléctricas desde el Arduino para encenderlo o apagarlo.
b. El botón es una entrada porque el Arduino necesita leer su estado (si está presionado o no). Sin pinMode(), el Arduino no sabría cómo interpretar la señal.

- Pregunta 8: ¿Qué podría causar que el LED no se encienda cuando se presiona el botón, aunque el código sea correcto?

a. Conexiones incorrectas en el circuito: Verifica que el LED esté correctamente conectado al pin digital y a GND.
b. Resistor quemado o LED dañado: Asegúrate de que el LED y el resistor no estén dañados.
c. Problemas en la placa Arduino: Comprueba que la placa Arduino esté conectada correctamente a la fuente de alimentación y que esté funcionando.

3. CONCLUSIONES

1. Conceptos de entradas y salidas digitales:

Los estudiantes han aprendido la diferencia entre entradas (como el botón) y salidas (como el LED) en el contexto de sistemas de control. Han visto cómo se configuran y controlan a través del código de Arduino, utilizando funciones como pinMode(), digitalRead() y digitalWrite().

2. Importancia de las resistencias:

Se ha comprendido la necesidad de las resistencias en el circuito, tanto para limitar la corriente del LED como para estabilizar el comportamiento del botón mediante una resistencia de pull-down. Este conocimiento es esencial para evitar daños a los componentes electrónicos y asegurar el correcto funcionamiento de los mismos.

3. Simulación y montaje del circuito:

La práctica en simuladores como Tinkercad permite a los estudiantes experimentar de forma virtual antes de implementar el circuito físico, lo que ayuda a reducir errores y aumentar la comprensión. El montaje en la protoboard refuerza el aprendizaje práctico y les introduce al prototipado rápido.

4. Desarrollo de habilidades de programación:

A través del ejercicio, los estudiantes han dado sus primeros pasos en la programación de microcontroladores. Han escrito, entendido y ejecutado un código que controla un LED a través de un botón, y han explorado mejoras como la implementación de un sistema de alternancia (toggle).

5. Análisis de problemas y solución de errores:

Se ha promovido el desarrollo de habilidades de resolución de problemas. A través de preguntas dirigidas y la experimentación con el circuito y el código, los estudiantes aprenden a diagnosticar y corregir errores comunes, como conexiones incorrectas o malinterpretaciones del código.