Control programado con Arduino

Práctica 7: El Semáforo Interactivo (Petición de Paso)

Hasta ahora hemos programado sistemas que funcionaban de forma aislada: o bien respondían inmediatamente a un botón, o bien seguían una secuencia fija en el tiempo. Hoy uniremos ambos mundos para diseñar un semáforo interactivo. Por defecto, los vehículos tendrán siempre vía libre (luz verde continua), pero cuando un peatón presione el pulsador, el Arduino interrumpirá el flujo normal para detener el tráfico y permitir un cruce seguro.

Objetivos de Aprendizaje

Al finalizar esta sesión, serás capaz de:

- Integrar componentes de entrada (pulsador) y de salida (LEDs) en un mismo sistema coordinado.
- Diseñar un algoritmo basado en estados donde el programa "espera de forma activa" una acción del usuario.
- Analizar críticamente el comportamiento de un microcontrolador cuando se combinan lecturas de sensores con pausas de tiempo de larga duración (`delay`).
- Desarrollar soluciones lógicas para simular prioridades de paso en entornos automatizados urbanos.

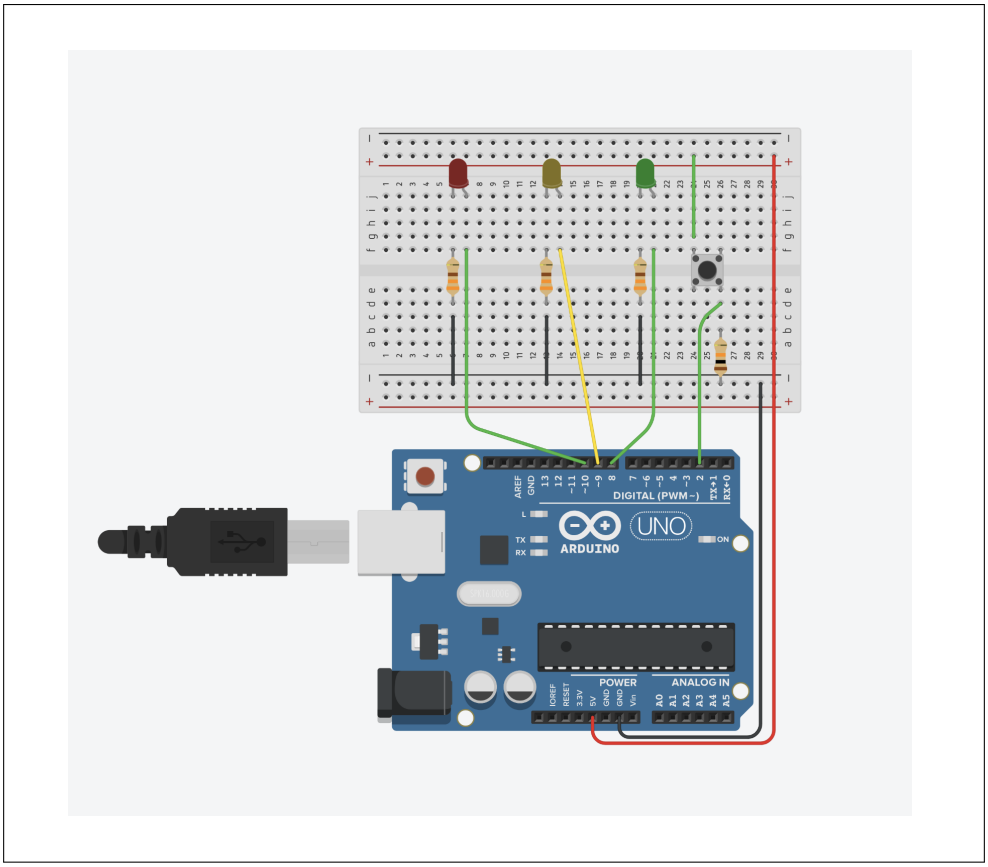
Componentes Necesarios

Busca y coloca los siguientes componentes en tu mesa de trabajo de TinkerCad:

- 1 Placa Arduino Uno.
- 1 Placa de pruebas pequeña (Protoboard).
- 3 Diodos LED (1 Rojo, 1 Amarillo, 1 Verde).
- 3 Resistencias de $220\,\Omega$ (para proteger los LEDs).
- 1 Pulsador de cuatro patillas.
- 1 Resistencia de $10\,\text{k}\Omega$ (para la configuración Pull-Down del pulsador).
- Cables de conexión virtuales.

Esquema de Montaje en la Protoboard

Este circuito es la combinación exacta de los montajes de la Práctica 3 y la Práctica 5. ¡Asegúrate de cablear con orden para evitar cruces!



Código Base: Espera Activa del Flujo

Copia el siguiente programa en el editor de TinkerCad. Observa que en la función `loop` no hay ningún retardo de tiempo inicial; el Arduino se limita a vigilar el botón a la máxima velocidad posible.

```
// Definicion de pines de salida para los LEDs
int ledRojo = 10;
int ledAmarillo = 9;
int ledVerde = 8;

// Definicion del pin de entrada para el boton
int botonPeaton = 2;

void setup() {
  pinMode(ledRojo, OUTPUT);
  pinMode(ledAmarillo, OUTPUT);
  pinMode(ledVerde, OUTPUT);
  pinMode(botonPeaton, INPUT);

  // Estado inicial por defecto: via libre para los coches
  digitalWrite(ledVerde, HIGH);
  digitalWrite(ledAmarillo, LOW);
  digitalWrite(ledRojo, LOW);
}

void loop() {
  // Leemos continuamente el estado del boton
  int peticionPaso = digitalRead(botonPeaton);

  // Si un peaton presiona el boton, iniciamos la secuencia de parada
  if (peticionPaso == HIGH) {
    delay(1000); // Pequeño tiempo de cortesia antes de reaccionar

    // 1. Pasar de Verde a Amarillo
    digitalWrite(ledVerde, LOW);
    digitalWrite(ledAmarillo, HIGH);
    delay(2000); // 2 segundos en amarillo de advertencia

    // 2. Pasar de Amarillo a Rojo (Coches detenidos / Peatones cruzan)
    digitalWrite(ledAmarillo, LOW);
    digitalWrite(ledRojo, HIGH);
    delay(5000); // 5 segundos para que los peatones crucen seguros

    // 3. Volver al estado de via libre (Coches avanzan)
    digitalWrite(ledRojo, LOW);
    digitalWrite(ledVerde, HIGH);
  }
}
```

Listing 1: Código para un semáforo interactivo con petición de paso.

¿Cómo funciona el código?

En este programa, el secreto está en la estructura de la función `loop` :

- **Espera sin bloqueos**: Mientras nadie pulse el botón, el microcontrolador lee la línea `int digitalRead` , comprueba que da `LOW`, ignora el bloque `if` y vuelve a empezar el bucle. Esto sucede miles de veces por segundo, por lo que el LED verde se mantiene encendido sin pausa alguna.
- **El problema de la insensibilidad temporal**: Una vez que el peatón pulsa el botón y entramos dentro del bloque `if`, el flujo secuencial toma el control y se ejecutan las órdenes `delay` y `delay` . Durante esos 7 segundos totales, el Arduino está "congelado" ejecutando las pausas. Si otro peatón volviera a pulsar el botón en ese intervalo, el Arduino no se enteraría, ya que no puede ejecutar la línea `digitalRead` mientras esté dormido por un `delay` .

El Reto de la Práctica 7

Pon a prueba tus habilidades resolviendo estas dos mejoras lógicas de ingeniería urbana:

1. **Tiempo de Cortesía de Tráfico (Cool-Down)**: En una ciudad real, si un peatón pulsa el botón, cruza y, justo al terminar, otro peatón vuelve a pulsar el botón, el tráfico se colapsaría. Modifica el código para añadir un tiempo de espera obligatorio de **5 segundos** al final del ciclo (después de volver a encender el verde) durante el cual el sistema ignore por completo cualquier pulsación del botón.
2. **Aviso acústico o visual de fin de paso**: Añade un parpadeo rápido en el LED rojo durante el último segundo de su activación para avisar de forma visual a los peatones que su tiempo de paso está a punto de agotarse.
3. *Pregunta para reflexionar*: ¿Cómo resolverías el problema de la insensibilidad del botón sin recurrir a hardware complejo? Si dividieras un retraso largo como `delay(5000)` en pequeños pasos de `delay(100)`, ¿cómo podría ayudar esto a comprobar el estado del botón más a menudo?