PRÁCTICA DE ARDUINO - LUCES INTERMITENTES (BLINKING LEDS)

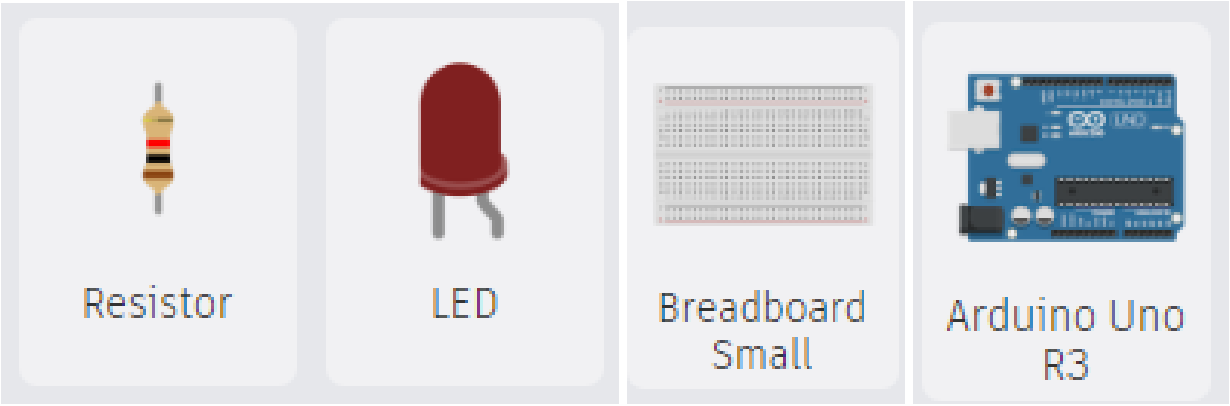
NIVEL: 1
DURACIÓN: 2 horas
OBJETIVO: Los estudiantes aprenderán a controlar varios LEDs para que parpadeen en un patrón definido
MATERIALES

- Placa Arduino Uno (o similar)
- 4 LEDs de diferentes colores
- 4 resistencias de 220 ohmios
- Protoboard
- Cables de conexión
- Computadora con el software Arduino IDE instalado
- ThinkerCad

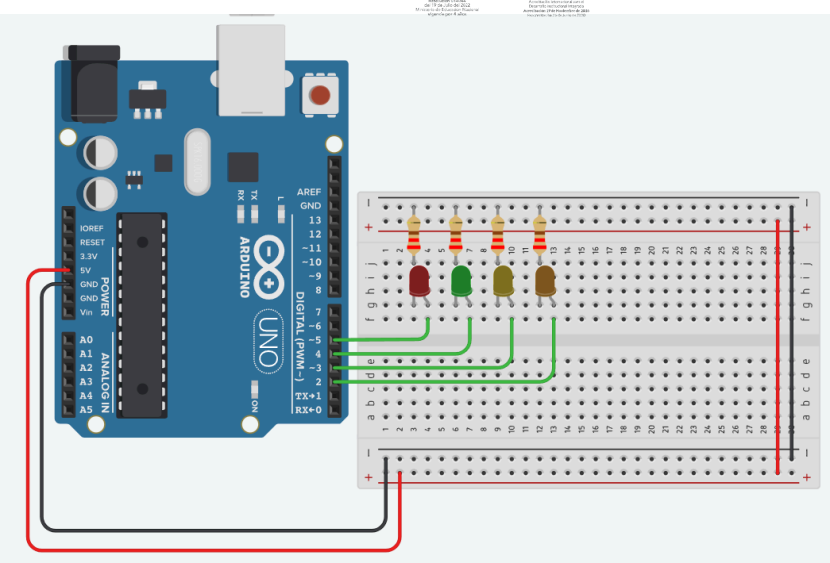
1. DESARROLLO DE LA PRÁCTICA

1.1 SIMULACIÓN EN THINKERCAD

1.1.1 COMPONENTES



1.1.2 MONTAJE



2.2 PROGRAMACIÓN EN ARDUINO IDE

Luces intermitentes en secuencia

```
// Definir los pines de los LEDs
const int led1 = 2;
const int led2 = 3;
const int led3 = 4;
const int led4 = 5;

void setup() {
  // Configurar los pines como salidas
  pinMode(led1, OUTPUT);
  pinMode(led2, OUTPUT);
  pinMode(led3, OUTPUT);
  pinMode(led4, OUTPUT);
}

void loop() {
  // Encender los LEDs en secuencia
  digitalWrite(led1, HIGH);// Encender LED 1
  delay(500);// Esperar 500 ms
  digitalWrite(led1, LOW);// Apagar LED 1

  digitalWrite(led2, HIGH);// Encender LED 2
  delay(500);// Esperar 500 ms
  digitalWrite(led2, LOW);// Apagar LED 2

  digitalWrite(led3, HIGH);// Encender LED 3
  delay(500);// Esperar 500 ms
  digitalWrite(led3, LOW);// Apagar LED 3

  digitalWrite(led4, HIGH);// Encender LED 4
  delay(500);// Esperar 500 ms
  digitalWrite(led4, LOW);// Apagar LED 4
}
```

Explicación

- Cada LED está conectado a un pin digital del Arduino (pines 2, 3, 4 y 5), y esos pines se configuran como salidas utilizando la función pinMode().
- Usamos la función digitalWrite() para encender (HIGH) o apagar (LOW) cada LED.
- La secuencia se controla con la función delay(), que pausa la ejecución durante 500 milisegundos entre cada cambio de estado (encendido y apagado).

Cambiando la Velocidad y la Secuencia: Parpadeo alternado rápido

```
// Definir los pines de los LEDs
const int led1 = 2;
const int led2 = 3;
const int led3 = 4;
const int led4 = 5;

void setup() {
  // Configurar los pines como salidas
  pinMode(led1, OUTPUT);
  pinMode(led2, OUTPUT);
  pinMode(led3, OUTPUT);
  pinMode(led4, OUTPUT);
}
void loop() {
  // Encender LEDs 1 y 3 al mismo tiempo
  digitalWrite(led1, HIGH);
  digitalWrite(led3, HIGH);
  delay(250);                 // Esperar 250 ms
  digitalWrite(led1, LOW);
  digitalWrite(led3, LOW);

  // Encender LEDs 2 y 4 al mismo tiempo
  digitalWrite(led2, HIGH);
  digitalWrite(led4, HIGH);
  delay(250);                 // Esperar 250 ms
  digitalWrite(led2, LOW);
  digitalWrite(led4, LOW);
}
```

Cambiando la Velocidad y la Secuencia: Parpadeo simultáneo

```
// Definir los pines de los LEDs
const int led1 = 2;
const int led2 = 3;
const int led3 = 4;
const int led4 = 5;

void setup() {
  // Configurar los pines como salidas
  pinMode(led1, OUTPUT);
  pinMode(led2, OUTPUT);
  pinMode(led3, OUTPUT);
  pinMode(led4, OUTPUT);
}
void loop() {
  // Encender todos los LEDs al mismo tiempo
  digitalWrite(led1, HIGH);
  digitalWrite(led2, HIGH);
  digitalWrite(led3, HIGH);
  digitalWrite(led4, HIGH);
  delay(100);                 // Esperar 100 ms

  // Apagar todos los LEDs
  digitalWrite(led1, LOW);
  digitalWrite(led2, LOW);
  digitalWrite(led3, LOW);
  digitalWrite(led4, LOW);
  delay(100);                 // Esperar 100 ms
}
```

2. ANÁLISIS DE RESULTADOS

- ¿Cómo afecta el valor del delay() al parpadeo de los LEDs?

El valor de delay() define la duración que los LEDs permanecen encendidos o apagados. Un valor más alto hará que el parpadeo sea más lento, mientras que un valor más bajo hará que el parpadeo sea más rápido. Puedes experimentar con diferentes valores de delay() para crear distintos efectos visuales.

- ¿Qué sucede si eliminamos el delay()?

Si eliminamos el delay(), los LEDs cambiarán de estado tan rápidamente que no podrás observar el parpadeo, ya que el programa se ejecuta a gran velocidad. Es probable que los LEDs parezcan estar encendidos continuamente. Para evitar esto, es importante usar delay() o alguna técnica de temporización como millis() para controlar el tiempo entre los cambios de estado.

- ¿Por qué es importante usar resistencias con los LEDs?

Las resistencias limitan la cantidad de corriente que pasa a través de los LEDs. Si no se usan resistencias, los LEDs podrían recibir demasiada corriente, lo que podría quemarlos y dañar los pines de salida del Arduino. En este caso, una resistencia de 220 ohmios es adecuada para proteger los LEDs y asegurar que funcionen correctamente sin sobrecargarse.

- ¿Qué sucede si conectamos más LEDs al Arduino? ¿Qué hay que tener en cuenta?

Arduino puede controlar más LEDs, pero es importante tener en cuenta que cada pin digital tiene un límite de corriente (normalmente 20 mA por pin). Si conectas demasiados LEDs directamente a los pines del Arduino sin precaución, podrías exceder el límite de corriente total que el Arduino puede proporcionar, lo que podría dañar la placa.

- ¿Cómo podríamos mejorar el código para que sea más eficiente?

Respuesta: En lugar de usar delay(), que detiene todo el código mientras espera, se podría utilizar la función millis() para controlar el tiempo sin pausar la ejecución del programa. Esto permitiría realizar otras tareas mientras los LEDs parpadean, como leer sensores o controlar otros dispositivos, haciendo el código más flexible y eficiente para proyectos más complejos.

- ¿Cómo podemos cambiar el brillo de los LEDs en lugar de solo encenderlos y apagarlos?

Para cambiar el brillo de los LEDs, podemos usar la función analogWrite() en pines que soporten PWM (Modulación por Ancho de Pulso). Esto permite ajustar el nivel de brillo del LED en lugar de simplemente encenderlo o apagarlo. El valor de analogWrite() varía entre 0 (apagado) y 255 (máximo brillo). Por ejemplo:

```
analogWrite(led1, 128); // Brillo medio
```

- ¿Qué sucede si conectamos los LEDs en paralelo en lugar de en serie?

Si conectas los LEDs en paralelo, cada LED recibirá el mismo voltaje, y cada uno funcionará de manera independiente. Esto es más eficiente cuando cada LED necesita su propia resistencia para limitar la corriente de forma adecuada. Si los conectas en serie, el voltaje se divide entre los LEDs, lo que puede no ser suficiente para encenderlos correctamente, además de que un fallo en un LED afectará a los demás.

3. CONCLUSIONES

a. Este proyecto ha proporcionado una excelente introducción al control de salidas digitales con Arduino, lo cual es fundamental para una amplia variedad de proyectos. El control de LEDs, aunque básico, enseña los principios fundamentales de cómo controlar dispositivos electrónicos mediante programación y cómo interactuar con componentes de hardware a través de los pines de salida del Arduino.

b. Durante el proyecto, aprendimos la importancia de utilizar resistencias para proteger tanto los LEDs como la placa Arduino de posibles daños debido al exceso de corriente.

c. Aunque la función delay() es útil para implementar parpadeos simples, también vimos que tiene sus limitaciones, ya que bloquea la ejecución del código durante el tiempo de espera. Para proyectos más complejos donde se necesite realizar múltiples tareas al mismo tiempo (como monitorear sensores mientras los LEDs parpadean), sería más eficiente utilizar técnicas de temporización no bloqueante como la función millis().

d. Este proyecto básico de luces intermitentes puede expandirse fácilmente para incluir más LEDs, variar el brillo mediante PWM, o agregar controles como botones o sensores para interactuar con los LEDs. Además, este concepto se puede aplicar en proyectos más grandes, como sistemas de iluminación, señales luminosas, y otros dispositivos interactivos controlados por Arduino.