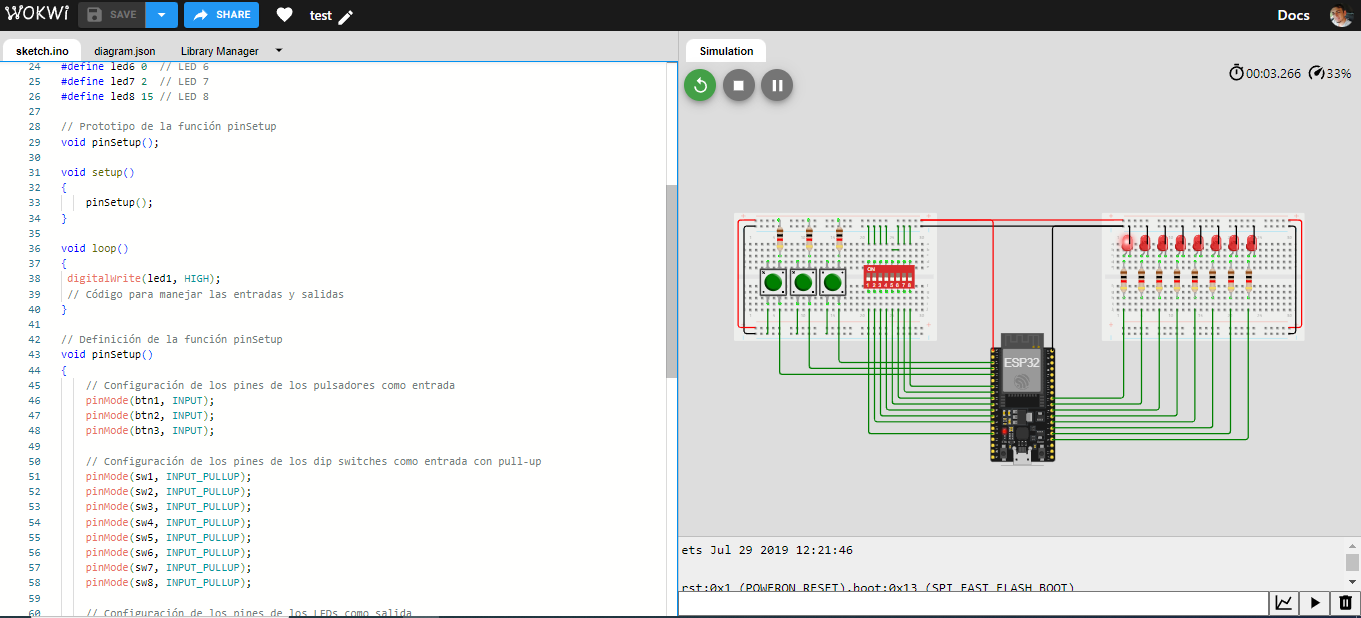
**Ejercicio 1: Encender un LED**

• Enciende el led1 conectado al GPIO18 de forma continua.



**Codigo:**

void loop()

{

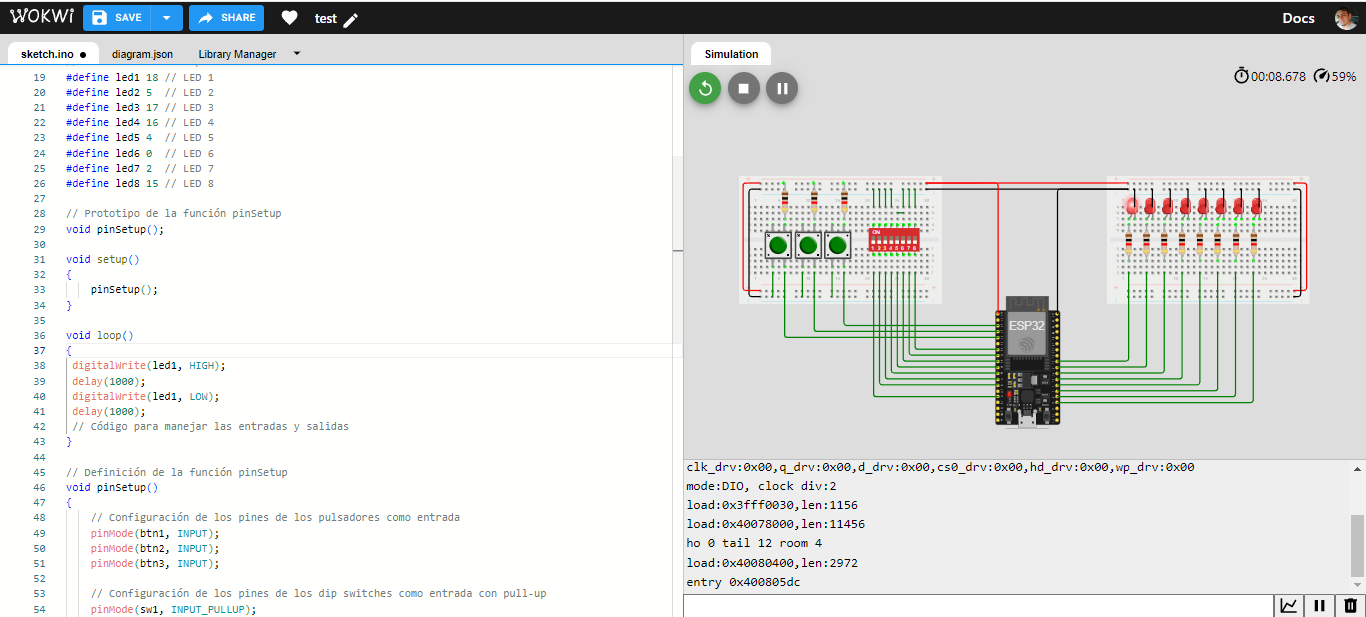
   digitalWrite(led1, HIGH);

   // Código para manejar las entradas y salidas

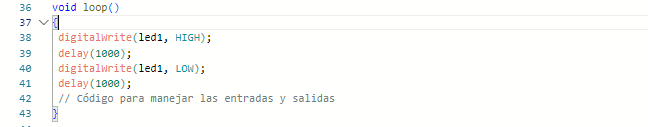
}

**Ejercicio 2: Parpadeo de un LED**

• Programa el led1 para que parpadee con un intervalo de 1 segundo.

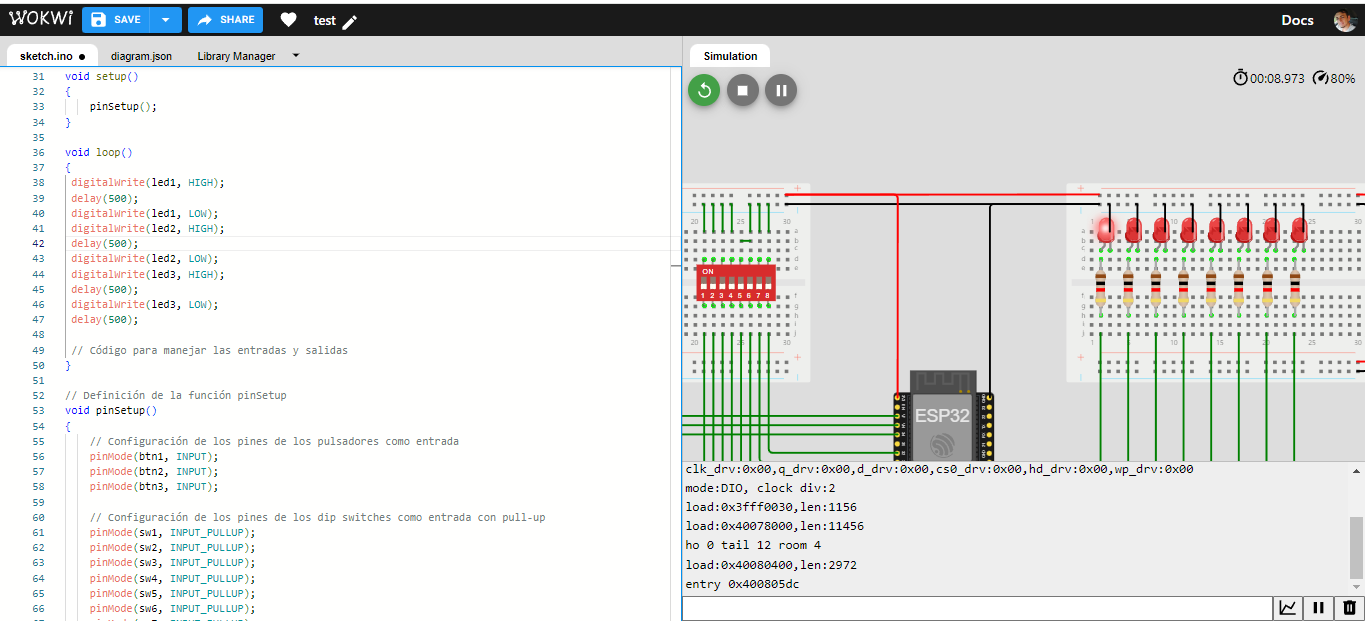


**Codigo:**

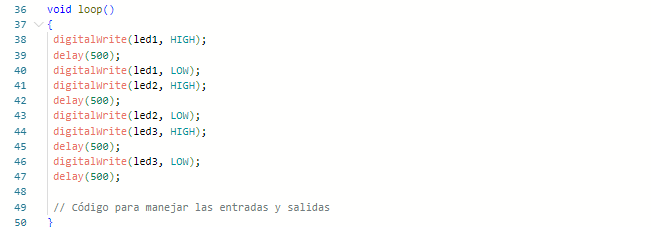


**Ejercicio 3: Secuencia de LEDs**

• Crea una secuencia que encienda los LEDs del led1 al led3 de forma sucesiva, cada uno durante 500ms.

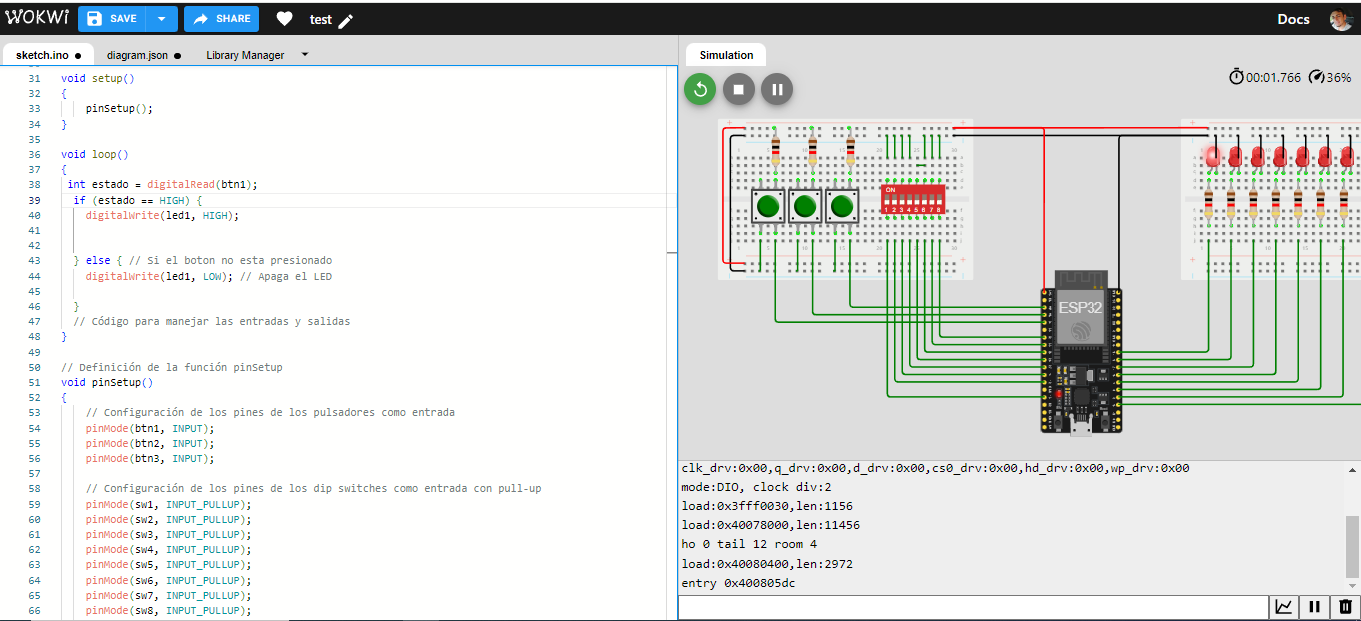


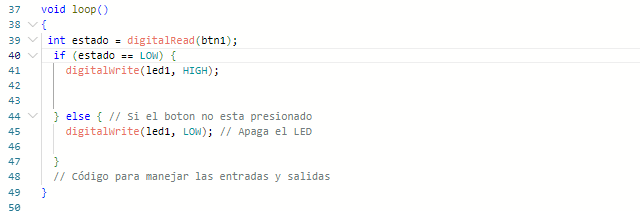
**Codigo:**



**Ejercicio 4: Control de LED con botón**

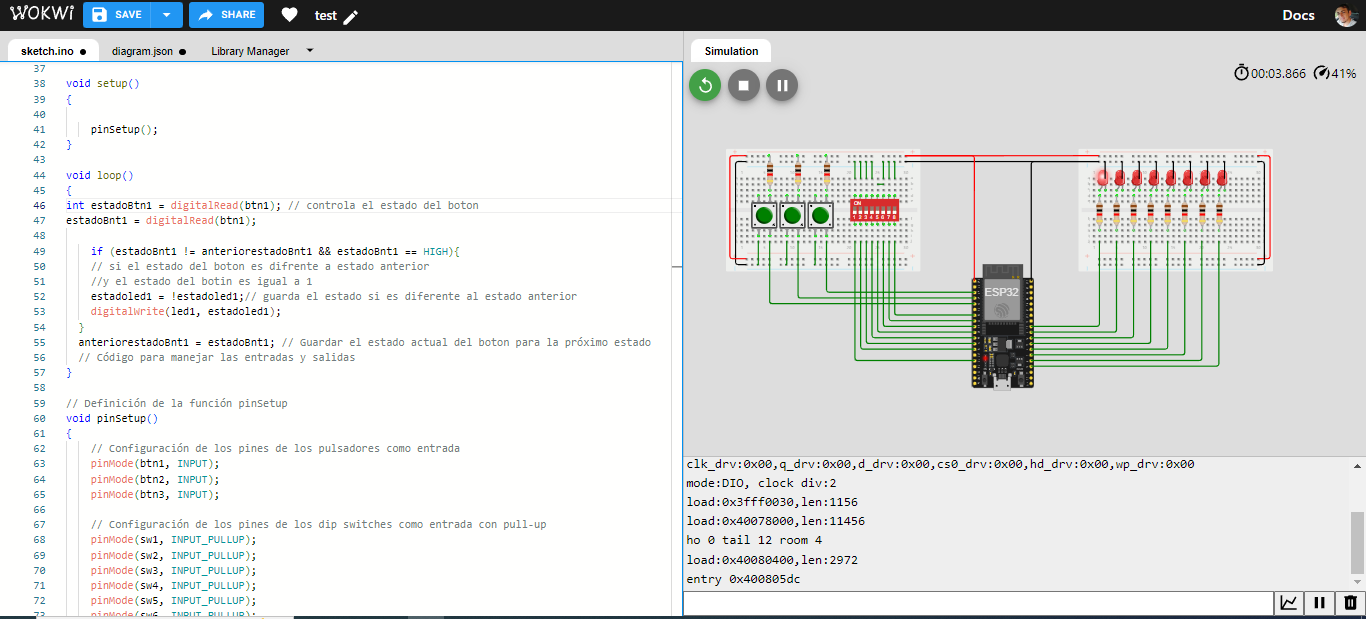
• Usa el btn1 para encender el led1 mientras se mantenga presionado.



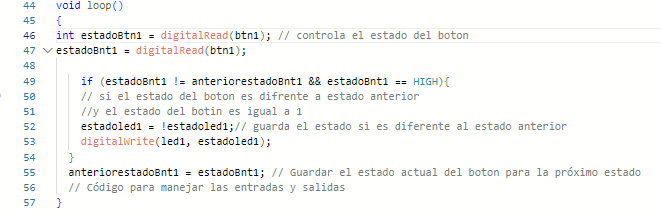
**Codigo:  
  
**

**Ejercicio 5: Uso de botón con estado**

• Cambia el estado del led1 cada vez que se presione y suelte el btn1.

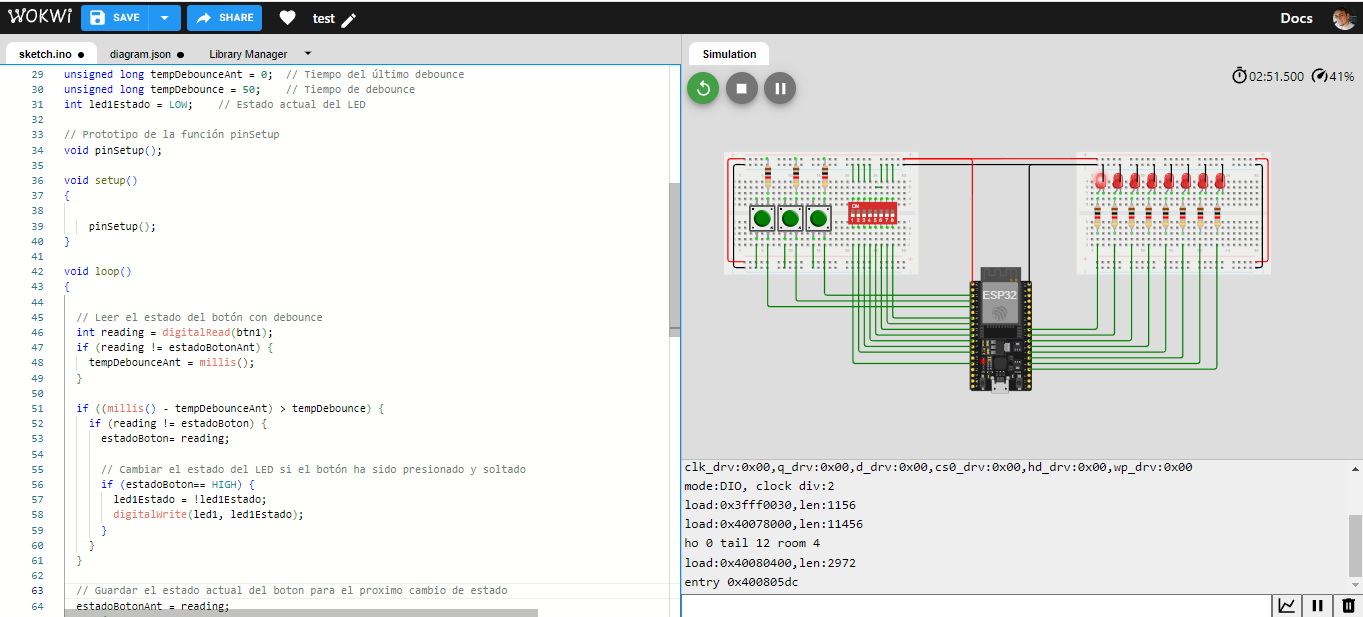


**Codigo:**

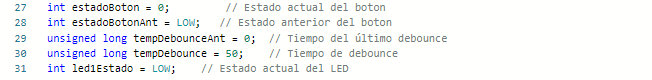


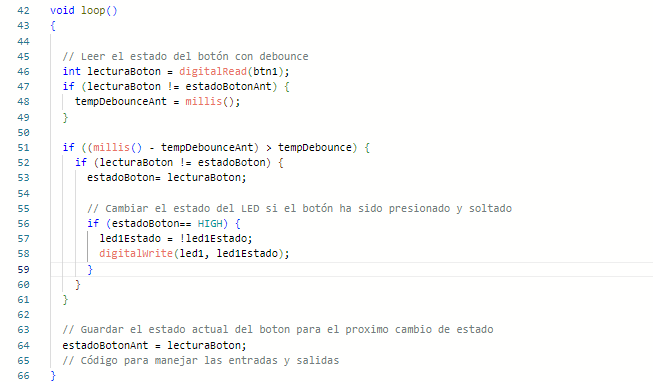
**Ejercicio 6: Debounce de botón**

• Implementa una lógica de debounce en el btn1 para evitar lecturas erróneas.



**Codigo:**





solucion por software.

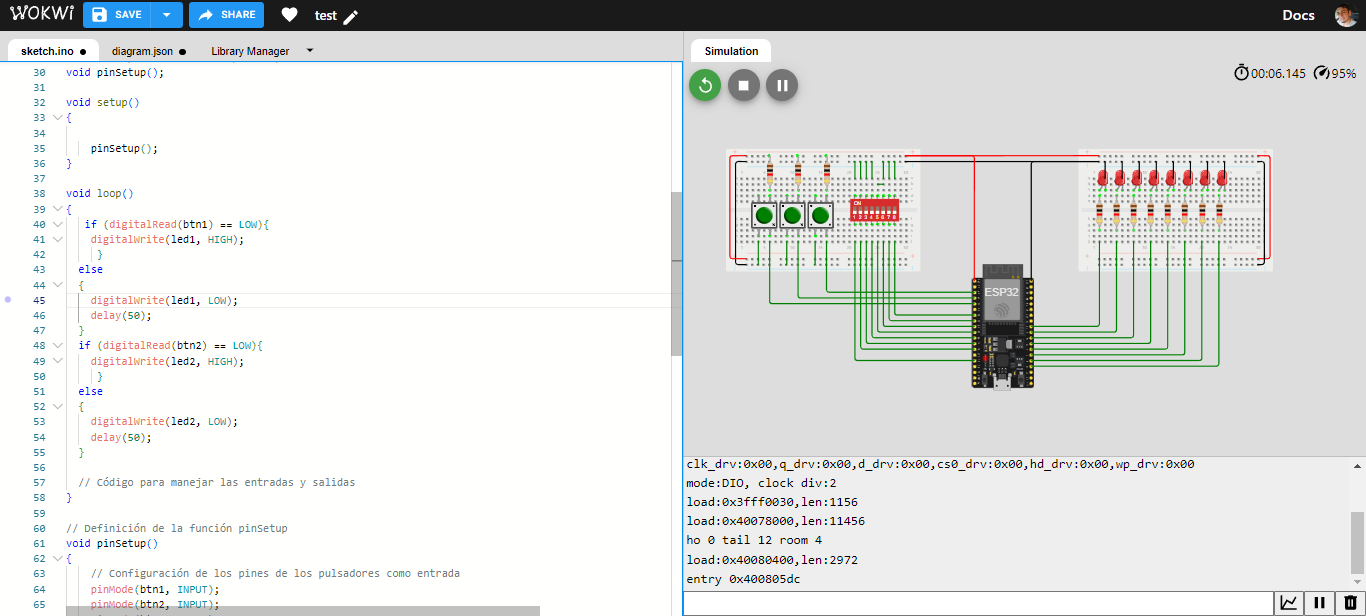
El debounce es una técnica utilizada en ingeniería de interfaces para eliminar las falsas pulsaciones que se producen en los dispositivos electrónicos, especialmente en los interruptores y botones. En Arduino, el proceso de eliminación de este rebote se llama “debounce”.

También se puede solucionar por hardware.

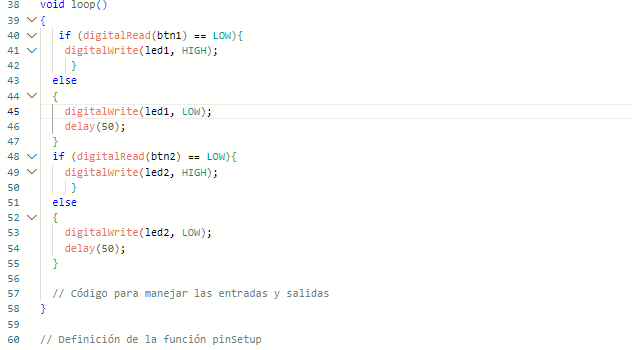
Para aplicar un debounce por hardware, se puede colocar un condensador en paralelo con el dispositivo. Un condensador de 1uF debería ser suficiente para filtrar la mayoría del ruido. Cuando se cierra el contacto, el condensador se descarga, y cuando se abre, absorbe el voltaje en el tiempo que se carga, eliminando los rebotes

**Ejercicio 7: Control de múltiples LEDs con botones**

• Usa btn1 y btn2 para controlar el estado de led1 y led2 respectivamente.

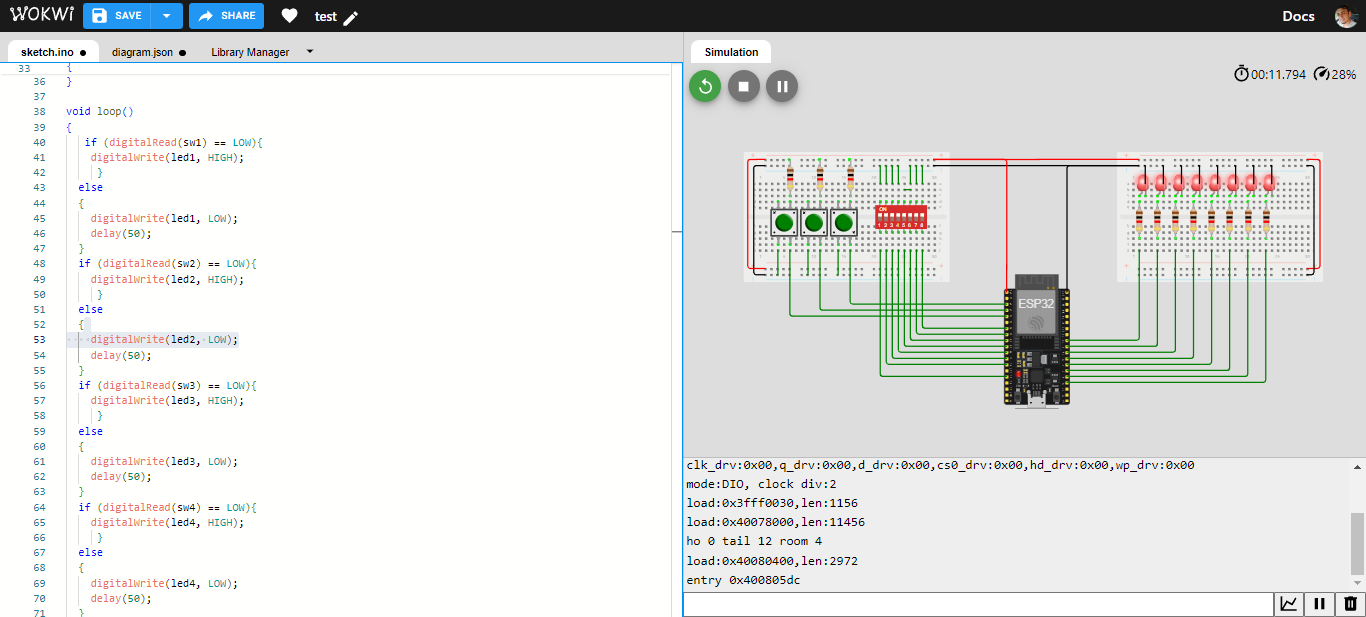


**Codigo:**



**Ejercicio 8: Uso de dip switches para control de LEDs**

• Lee el estado de los dip switches sw1.1 a sw1.8 y refleja el estado en los led1 a led8.



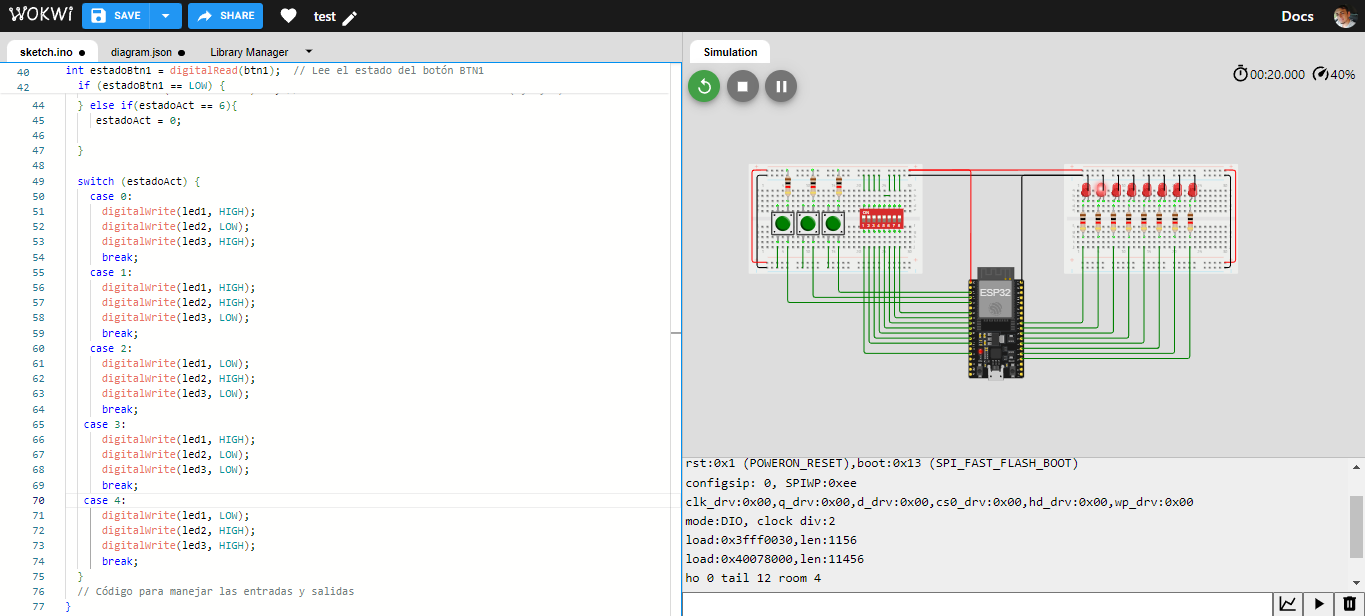
**Codigo:**

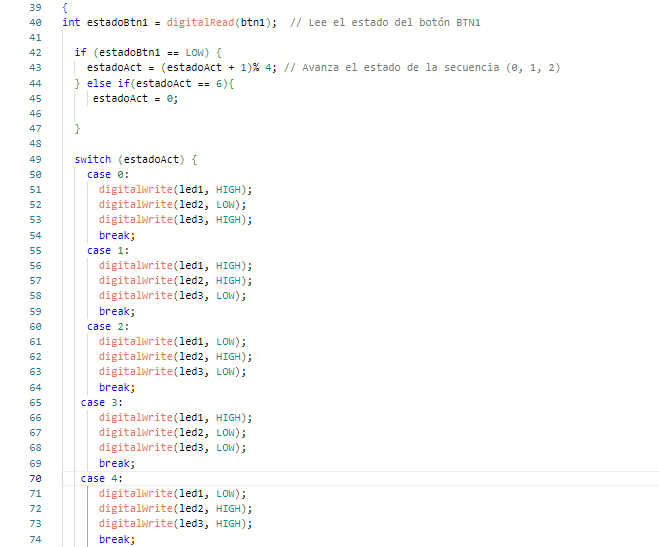




**Ejercicio 9: Secuencia de LEDs con botón**

• Crea una secuencia de luces que avance cada vez que se presione btn1.

**Codigo:**



**Investigacion:**estadoAct = (estadoAct + 1)% 4;: Esto es una operación que se ejecuta si la condición en la línea anterior es verdadera. estadoAct parece ser una variable que representa el estado actual en algún tipo de secuencia (probablemente la secuencia de luces en este caso). Esta línea de código incrementa estadoAct en 1, y luego toma el resto de la división de estadoAct entre 4.

• Por ejemplo, si estadoAct es 0, después de ejecutar esta línea, se convierte en 1.

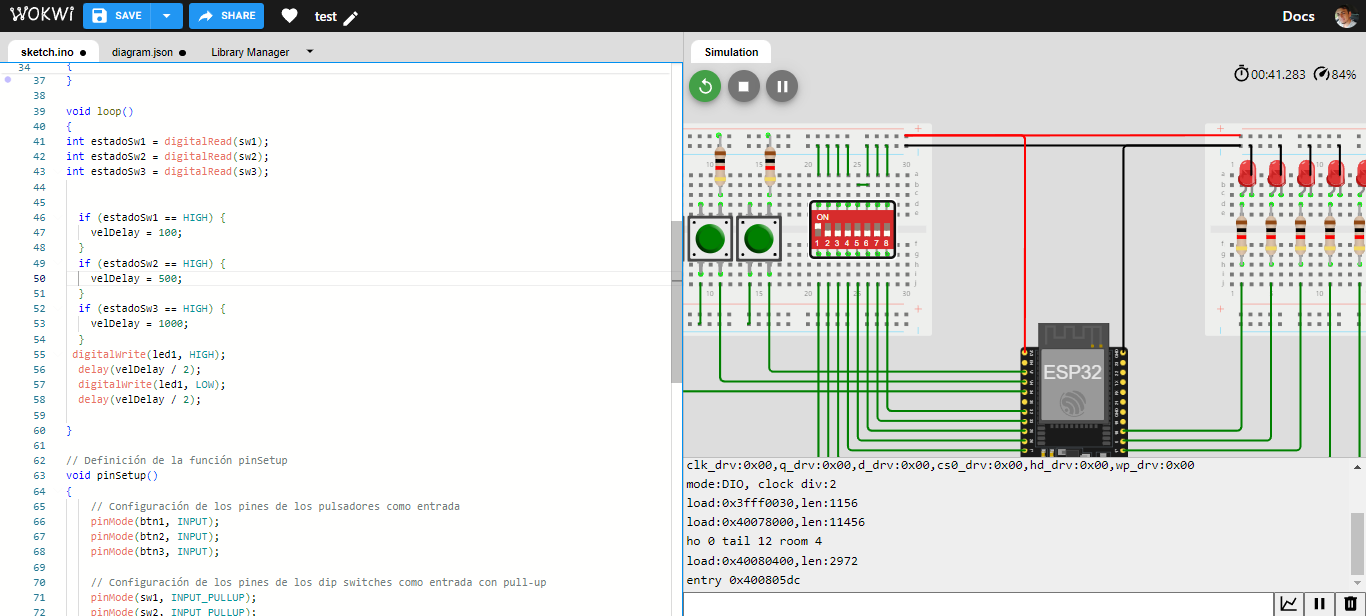
• Si estadoAct es 1, se convierte en 2.

• Si estadoAct es 2, se convierte en 3.

• Si estadoAct es 3, se convierte en 0 (debido al operador % 4).

**Ejercicio 10: Control de velocidad de parpadeo con dip switch**

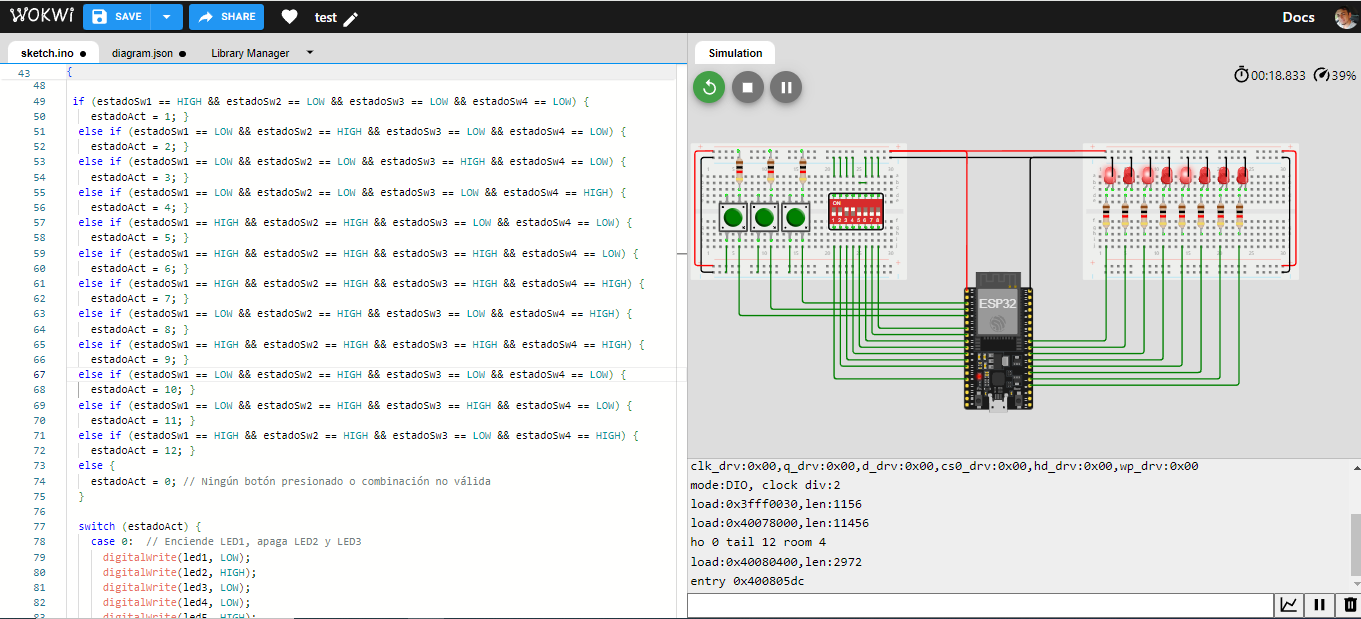
• Utiliza los dip switches sw1.1 a sw1.3 para controlar la velocidad de parpadeo de led1, asignando distintas velocidades.



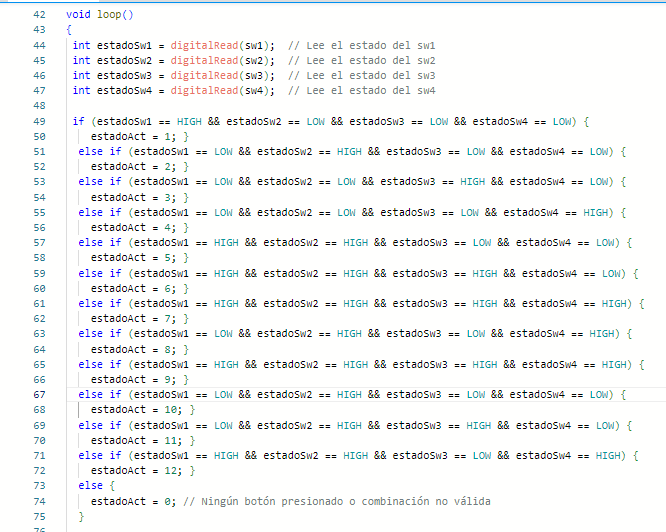
**Codigo:**

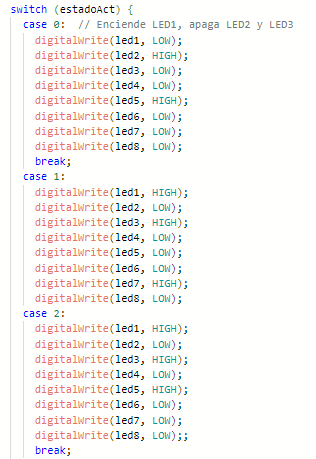
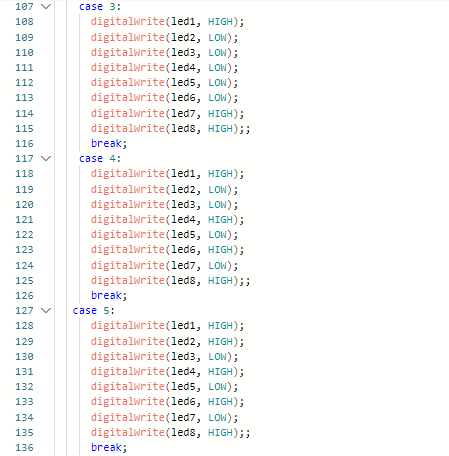
**Ejercicio 11: Patrón de parpadeo de LEDs con dip switches**

• Establece un patrón de parpadeo para los led1 a led8 basado en la combinación de estados de sw1.1 a sw1.4. Por ejemplo, cada posición activa del switch puede representar un patrón diferente (como parpadeo rápido, lento, secuencial, etc.).



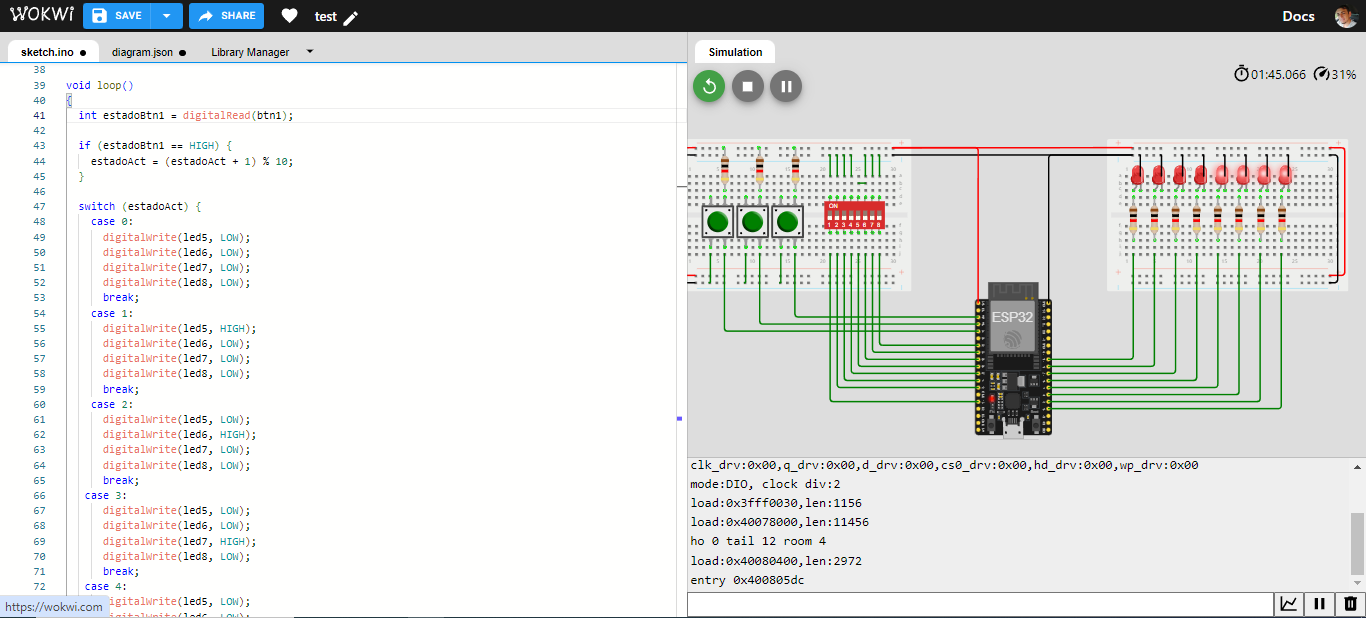
**Codigo:**



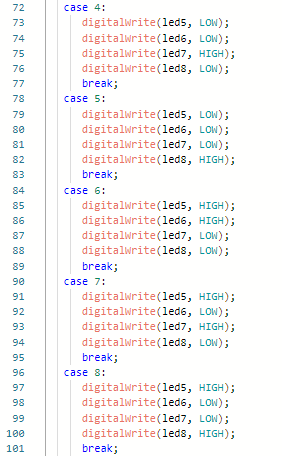


**Ejercicio 12: Medidor de pulsaciones**

• Programa un contador de pulsaciones utilizando btn1. El número de pulsaciones debe mostrarse en una secuencia de LEDs (por ejemplo, led5 a led8 donde cada LED representa una cantidad de pulsaciones).

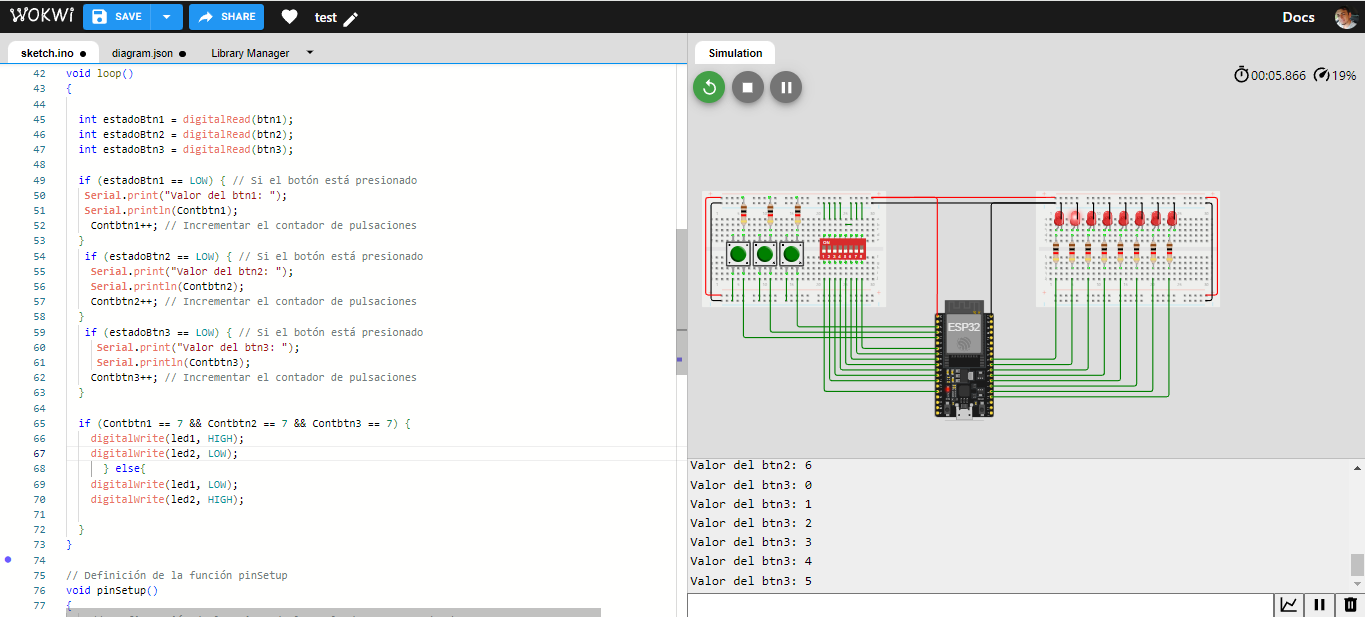


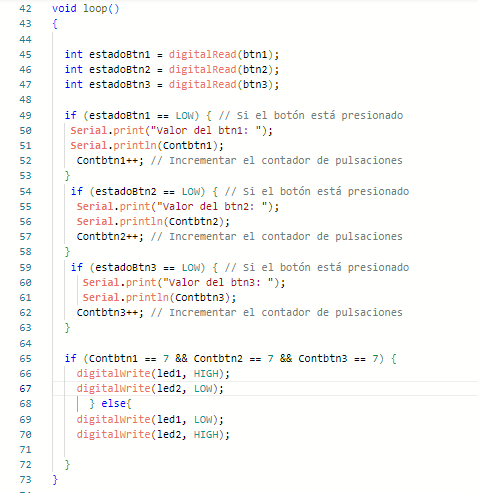
**Codigo:**

**Ejercicio 13: Contraseña con botones**

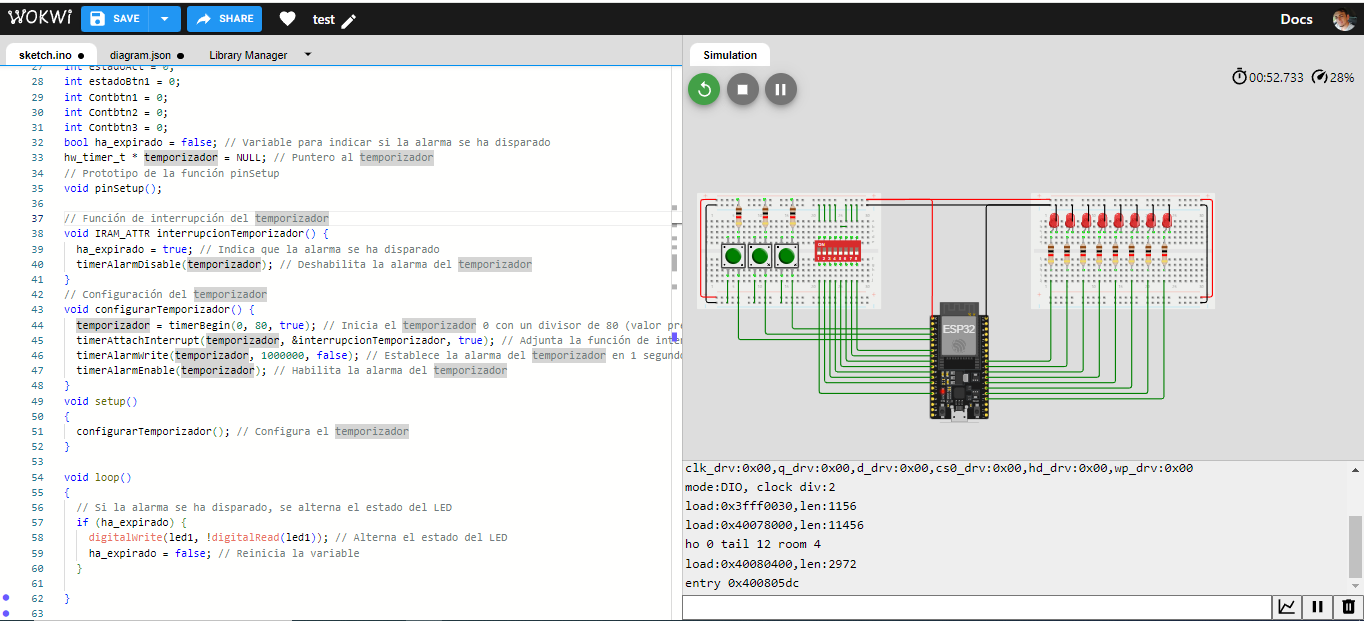
• Implementa un sistema de contraseña usando btn1, btn2, y btn3 donde una secuencia específica de pulsaciones activa led1. Si la secuencia es incorrecta, led2 debería encenderse.



**Codigo:**  


**Ejercicio 14: Aplicación de timers para control de LEDs**

• Utiliza el temporizador del ESP32 para controlar el parpadeo de led1 a led4 sin usar la función delay(), permitiendo que el programa ejecute otras tareas mientras los LEDs parpadean.



**Codigo:**

bool ha\_expirado = false; // Variable para indicar si la alarma se ha disparado

hw\_timer\_t \* temporizador = NULL; // Puntero al temporizador

// Prototipo de la función pinSetup

void pinSetup();

// Función de interrupción del temporizador

void IRAM\_ATTR interrupcionTemporizador() {

  ha\_expirado = true; // Indica que la alarma se ha disparado

  timerAlarmDisable(temporizador); // Deshabilita la alarma del temporizador

}

// Configuración del temporizador

void configurarTemporizador() {

  temporizador = timerBegin(0, 80, true); // Inicia el temporizador 0 con un divisor de 80 (valor predeterminado) y habilita la interrupción de nivel superior

  timerAttachInterrupt(temporizador, &interrupcionTemporizador, true); // Adjunta la función de interrupción al temporizador y habilita la interrupción de nivel superior

  timerAlarmWrite(temporizador, 1000000, false); // Establece la alarma del temporizador en 1 segundo (en microsegundos) y no se repite

  timerAlarmEnable(temporizador); // Habilita la alarma del temporizador

}

void setup()

{

  configurarTemporizador(); // Configura el temporizador

}

void loop()

{

  // Si la alarma se ha disparado, se alterna el estado del LED

  if (ha\_expirado) {

    digitalWrite(led1, !digitalRead(led1)); // Alterna el estado del LED

    ha\_expirado = false; // Reinicia la variable

  }

}

**investigación:  
Función configurarTemporizador():**

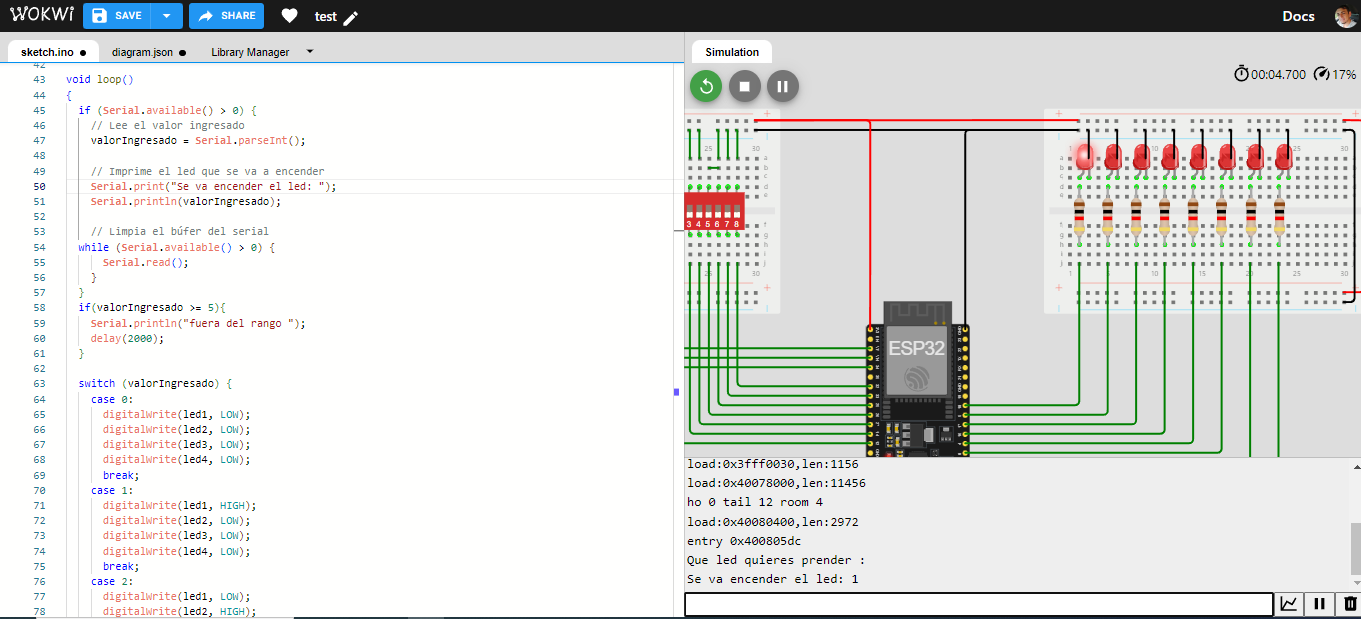
* **temporizador = timerBegin(0, 80, true);**: Esta línea inicia el temporizador 0. El primer parámetro (0) indica el número de identificador del temporizador. El segundo parámetro (80) es el divisor del temporizador, que determina la frecuencia de la señal de salida del temporizador. En este caso, la frecuencia será de 80 Hz. El tercer parámetro (true) habilita la interrupción de nivel superior para el temporizador.
* **timerAttachInterrupt(temporizador, &interrupcionTemporizador, true);**: Esta línea asocia la función interrupcionTemporizador con el temporizador 0 y habilita la interrupción de nivel superior. Cuando la alarma del temporizador se dispara, se llamará a la función interrupcionTemporizador.
* **timerAlarmWrite(temporizador, 1000000, false);**: Esta línea configura la alarma del temporizador para que se dispare después de 1 segundo. El primer parámetro (temporizador) es el identificador del temporizador. El segundo parámetro (1000000) es el tiempo de espera de la alarma en microsegundos. El tercer parámetro (false) indica que la alarma no se repetirá una vez que se dispare.
* **timerAlarmEnable(temporizador);**: Esta línea habilita la alarma del temporizador. Una vez habilitada, la alarma se disparará después del tiempo de espera configurado.

**Función interrupcionTemporizador():**

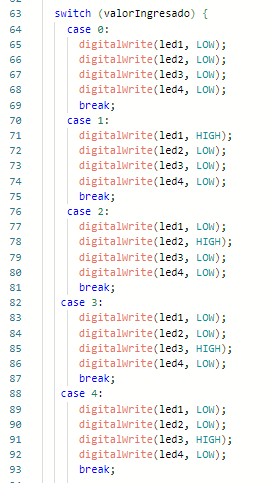
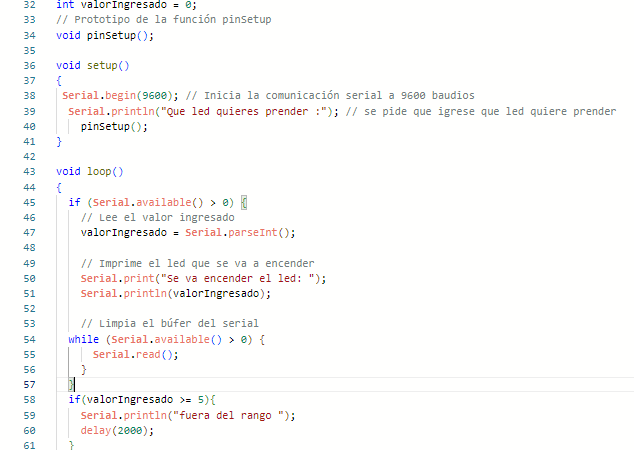
* **ha\_expirado = true;**: Esta línea establece la variable ha\_expirado en true. Esta variable se puede usar para indicar que la alarma del temporizador se ha disparado.
* **timerAlarmDisable(temporizador);**: Esta línea desactiva la alarma del temporizador. Esto evita que la alarma se dispare nuevamente.

**Ejercicio 15: Control de LEDs mediante comunicación serial**

• Escribe un programa que reciba comandos a través del puerto serie para controlar los LEDs. Por ejemplo, enviar '1' podría encender led1, '2' apagar led2, etc.

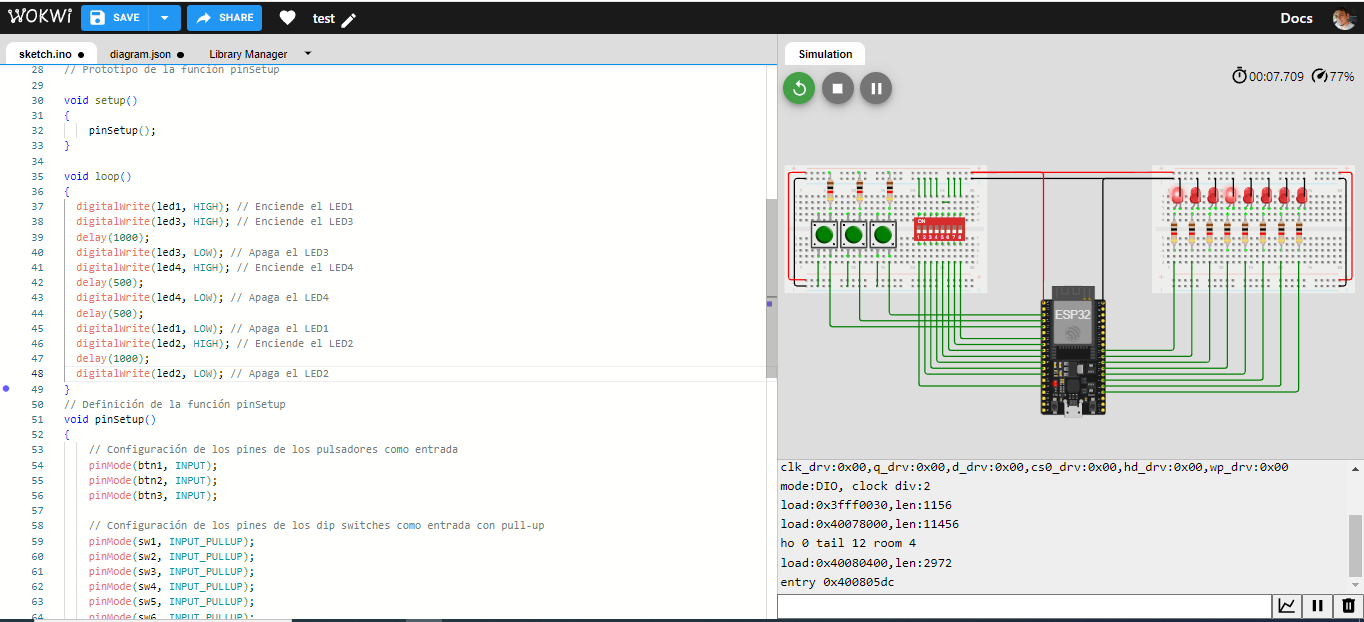


**Codigo:**



**Ejercicio 16: Secuencia de luces de emergencia**

• Simula luces de emergencia con los LEDs, donde led1 y led2 parpadean alternativamente en un patrón rápido, mientras que led3 y led4 lo hacen en un patrón más lento.



**Codigo:**

